PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F X

A2

(11) International Publication Number: WO 90/15379

(43) International Publication Date: 13 December 1990 (13.12.90)

(21) International Application Number: PCT/US90/02889

(22) International Filing Date: 15 May 1990 (15.05.90)

(30) Priority data:
90385 23 May 1989 (23.05.89) IL

(71)(72) Applicant and Inventor: ISAACSON, Joel [US/IL]; 9 Skolnick Street, 76 100 Rehovot (IL).

(74) Agents: GALLOWAY, Peter, D. et al.; Ladas & Parry, 26 West 61 Street, New York, NY 10023 (US).

(81) Designated States: AT (European patent), BE (European patent), CA, CH (European patent), DE (European patent)*, DK (European patent), ES (European patent), FR (European patent), GB (European patent), IT (European patent), JP, LU (European patent), NL (European patent), SE (European patent), US.

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: APPARATUS AND METHOD FOR IMAGE PROCESSION

(57) Abstract

A system for and methods of image processing are disclosed. The system comprises image acquisition apparatus for acquiring pixels of a digital image, permutation apparatus for receiving the pixels, for reordering pixels of the digital image into a plurality of abstracted images at a large reduction factor and for combining the abstracted images into many sets of abstracted images combinable into abstracted images at different small reduction factors and storage apparatus for receiving and for storing the abstracted sets of digital images. One method comprises an efficient way of rotating images stored as abstracted images. Other methods enable transformation between different storage formats.

DESIGNATIONS OF "DE"

Until further notice, any designation of "DE" in any international application whose international filing date is prior to October 3, 1990, shall have effect in the territory of the Federal Republic of Germany with the exception of the territory of the former German Democratic Republic.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MC	Monaco
AU	Australia	FI	Finland	MG	Madagascar
BB	Barbados	FR	France	ML	Mali
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Fasso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GR	Greece	NL	Netherlands
BJ	Benin	HU	Hungary	NO	Norway
BR	Brazil	IT	Italy	RO	Romania
CA	Canada	JP	Japan	SD	Sudan
CF	Central African Republic	KP	Democratic People's Republic of Korea	SE	Sweden
CG	Congo			SN	Senegal
CH	Switzerland	KR	Republic of Korea	SU	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
DE	Germany, Federal Republic of	LK	Sri Lanka	TG	Togo
DK	Denmark	LU	Luxembourg	US	United States of America

APPARATUS AND METHOD FOR IMAGE PROCESSION

FIELD OF THE INVENTION

The present invention relates to image storage methods generally and to permutations of pixel storage order in particular.

BACKGROUND OF THE INVENTION

Digitized images are rectangular arrays of pixel data which are usually acquired serially from a scanning device. Some color scanning devices present a complete pixel at a time, comprising three colors, typically cyan, yellow and magenta; other scanning devices present, at one time, a single row of partial pixels of one of the three colors, and alternate rows among the three colors.

It is known in the prior art to store digitized images in sequential format. The images are typically stored on magnetic disks, in sequentially written blocks of typically between 512 and 8192 bytes, where one block is the minimum number of bytes which can be read during one disk access. This block storage structure is only optimal when all the data accessed on each block is utilized. For data scattered in parts of many blocks, access becomes expensive in that many more disk reads are necessary to retrieve the desired information than would be needed if the desired data were clustered in complete blocks.

Computer memory is random access; sequential access and random access to computer memory each takes generally the same length of time which is typically much less than disk access time. Unfortunately, computer memory storage is much smaller than disk storage and thus, cannot be used to store and to access digitized images of many Mbytes.

Digitized images are usually displayed on Cathode Ray Tube (CRT) devices having a resolution of 75 - 100 pixels per inch. Since most scanners scan at a resolution of about 300 pixels per inch, displaying a digitized image requires a reduction of the volume of information by a factor typically varying between 9 and 16.

SUMMARY OF THE INVENTION

It is an object of the present invention to utilize the properties of both disk storage and computer memory to store digitized images.

Therefore, there is provided in accordance with the present invention, an image processing system comprising image acquisition apparatus for acquiring pixels of a digital image, permutation apparatus for receiving the pixels subsequent to their acquisition by the image acquisition apparatus and for ordering the pixels, thereby to reorder the digital image and storage apparatus for receiving and for storing the multiplicity of abstracted images.

Further, there is provided in accordance with an embodiment of the present invention an image processing system comprising an image acquisition apparatus for acquiring pixels of a digital image, permutation apparatus for receiving the pixels subsequent to their acquisition by the image acquisition apparatus and for ordering the pixels thereby to reorder the digital image into a multiplicity of abstracted images at a reduction factor and storage apparatus for receiving and for storing the multiplicity of abstracted images.

Additionally, there is provided in accordance with an embodiment of the present invention an image processing system comprising image acquisition apparatus for acquiring pixels of a digital image, permutation apparatus for receiving the pixels subsequent to their acquisition by the image acquisition apparatus, for ordering the pixels thereby to reorder the digital image into a plurality of abstracted images at a relatively large reduction factor and for combining the abstracted images into a multiplicity of sets of the abstracted images combinable into a plurality of abstracted images at a plurality of different relatively small reduction factors and storage apparatus for receiving and for storing the digital image in the multiplicity of sets.

Additionally, in accordance with any of the embodiments of the present invention, the permutation apparatus may comprise an apparatus for dividing the digital image into a plurality of bands comprising a predetermined number of rows of the digital image and apparatus for operating on the plurality of bands in a band by band manner. The apparatus for operating operates separately on each band.

Further, in accordance with any of the embodiments of the present invention, portions of the bands are written by the apparatus for operating to the storage apparatus when the portions of the band fill at least one file system block.

There is provided in accordance with the present invention, an image processing method including the steps of acquiring pixels of a digital image, receiving the pixels subsequent to their acquisition in the step of acquiring and ordering the pixels thereby to reorder the digital image into a multiplicity of abstracted images and storing the multiplicity of abstracted images.

There is provided in accordance with an embodiment of the present invention an image processing method including the steps of acquiring pixels of a digital image, receiving the pixels subsequent to their acquisition in the step of acquiring and ordering the pixels thereby to reorder the digital image into a plurality of abstracted images at a relatively large reduction factor and combining said abstracted image into a multiplicity of sets of the abstracted images combinable into a plurality of abstracted images at a plurality of different relatively small reduction factors and storing the digital image in the multiplicity of sets.

There is provided in accordance with an embodiment of the present invention, an image processing method including the steps of of sequentially acquiring pixels of a digital image, sequentially receiving the pixels subsequent to their acquisition in the step of sequentially acquiring and ordering the pixels thereby to reorder the digital image into a multiplicity of abstracted images at a reduction factor and of storing the multiplicity of abstracted images.

Moreover, there is provided in accordance with the present invention, an image transformation and rotation method including the steps of receiving an image stored in a permuted format, transforming the image in a permuted format to a tile format via an inverse transformation of the permuted format to a sequential format and via a forward transformation from the sequential format to the tile format, of rotating the tile format image by rotating the location of each tile of the tile format image, utilizing an amount of computer memory less than the image size and retransforming the rotated tile format image to the permuted format via an inverse transformation from the tile format to the sequential format and via a forward transformation from the sequential format to the permuted format.

Further, in accordance with the present invention, the step of rotation includes the steps of receiving and buffering a number of tiles of a source image to be rotated, of writing to a storage medium a different number of tiles of a destination image that are fully defined by rotations of the buffered tiles of a source image, and repeating the steps of receiving and writing until the source image no longer contains tiles.

There is provided in accordance with an embodiment of the present invention an image rotation method including the steps of receiving and buffering a number of tiles of a source image to be rotated and of writing to a storage medium a different number of tiles of a destination image that are fully defined by rotations of the buffered tiles of a source image, and repeating the steps of receiving and writing until the source image no longer contains tiles.

There is provided in accordance with the present invention, an image processing system comprising an image acquisition apparatus for acquiring pixels of a digital image, apparatus for dividing the digital image into a plurality of bands each comprising a predetermined number of rows, and permutation apparatus for operating on the plurality of bands in a band by band manner and for reordering the location of the pixels.

Further, there is provided in accordance with present invention, an image processing system comprising a storage apparatus for receiving and for storing the bands in a band by band manner.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood and appreciated more fully from the following detailed description taken in conjunction with the drawings in which:

Fig. 1 is a block diagram illustration of an image storage system using the image storage method of Fig. 2B;

Fig. 2A is a pictorial illustration of a sequentially stored image;

Fig. 2B is a pictorial illustration of a permutation based image storage method constructed and operative in accordance with the present invention;

Fig. 2C is a pictorial illustration of an alternative view of the permutation based image storage method of Fig. 2B.

Fig. 3 is a Venn diagram illustration of the regions of overlap between multiple abstraction levels useful in an alternate embodiment of Fig. 2;

Fig. 4 is a pictorial illustration of an order for storing the regions of overlap of Fig. 3;

Fig. 5 is a pictorial illustration of an alternative order for storing the regions of overlap of Fig. 3;

Fig. 6 is a pictorial illustration of a method for transforming an image stored in one format, or permutation, to an image stored in another format;

Fig. 7 is a pictorial illustration of a tile format permutation useful for rotating stored images;

Fig. 8 is a pictorial illustration of a method for rotating images 90^{o} using the tile format of Fig. 7;

Fig. 9 is a pictorial illustration of a method for rotating images 45^{o} using the tile format of Fig. 7; and

Fig. 10 is a pictorial illustration of a band permutation image storage method useful in the image storage method of Figs. 2.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Reference is now made to Fig. 1 which illustrates an image acquisition and storage system using the image storage method of the present invention. The system is based on a variation of standard direct memory address (DMA) techniques and comprises an image acquisition unit 110, typically a scanner or a CCD camera, for acquiring a digitized image and a scanner acquisition unit 112, similar in operation to a DMA controller, for receiving pixels of the digitized image, for permuting the addresses of the pixels within the digitized image according to the method of the present invention, and for transmitting the permuted pixels to temporary storage in a computer memory 114. The system additionally comprises a Central Processing Unit (CPU) 116, such as the 80386 microprocessor from Intel Corp., for directing the scanner acquisition unit 112. Since the computer memory 114 has a limited size, typically less than 10 Megabytes, which is generally significantly smaller than the size of the digitized image, typically in the tens of Megabytes, the memory 114 cannot hold the entirety of the permuted image. Thus, the permuted pixels temporarily stored in memory 114 are typically periodically transferred to a storage medium 118, via a storage medium DMA controller 119, in parallel with the entering of other permuted pixels into

memory **114**. The storage medium **118** is operative for generally longer term storage.

Scanner acquisition unit **112** permutes the addresses of the pixels according to a mapping transformation which typically is chosen to optimize frequent operations on the digitized image, such as reducing the size of the image in order to display it on a display device.

The mapping transformation is typically a permutation of the set $A=\{0,1,2,..N-1\}$ onto itself. A permutation P is denoted $i \rightarrow p_i$ where i is an element of the set of initial pixel addresses, p_i is a pixel address to which the ith initial pixel is permuted, known as a permuted pixel address, and N is the number of pixels in the image. Equation **1** presents a representation for the permutation P.

$$P = \begin{bmatrix} 0 & 1 & 2 & \cdots & N-1 \\ p_0 & p_1 & p_2 & \cdots & p_{N-1} \end{bmatrix} \tag{1}$$

The upper row of the notation typically indicates the initial pixel addresses i and is denoted $T(P)$; the lower row of the notation typically indicates the permuted pixel addresses p_i. The notation specifies that any untransformed pixels i (i.e. any pixels for which $p_i=i$) in the permutation P are not included in the representation, but they are present in the mapping. Moreover, a permutation P operates on the entire digitized image.

It will be understood that the permutational transformations are invertible and that they do not perform any compression or expansion of the image.

The smallest set of sequential pixel addresses, known as an interval, that contains $T(P)$ is typically called P's band, $b(P)$. For example, let C be the transformation given in equation **2**.

$$C = \begin{bmatrix} 20 & 35 \\ 35 & 20 \end{bmatrix} \tag{2}$$

$T(C)$ accordingly contains the addresses {20,35} and the band $b(C)$ is the interval [20,...,35]. A further example of a band $b(P)$ is shown in Fig. **10** which illustrates a digitized image, denoted **120**, organized into bands, denoted **122a-e**. Each band **122**, as mentioned hereinabove, is typically comprised of a predetermined interval of pixel addresses of the image **120**. Moreover, the bands **122** do not overlap with each other.

Thus, it is possible to define non-overlapping (i.e. having disjoint sets $b(P_j)$) transformations P_j whose bands, for the example shown in Fig. **10**, are **122a-e**. It will be appreciated that the number of non-overlapping transformations depends on the number of bands **122** chosen and is typically denoted by n_b. A transformation P for the entire image **120** is the product of the non-overlapping transformations P_j as defined in equation **3**.

$$P = P_1 \cdot P_2 \cdots P_{n_b}. \tag{3}$$

The transformations P_j are typically related to the transformation P_1 by the addition of a base value c_j. If P_1 is defined by $i \rightarrow p_{1i}$, then P_j is defined by

$$i+c_j \rightarrow p_{1i}+c_j \tag{4}$$

In other words, each transformation P_j is the transformation P_1 shifted by the base value c_j.

For example, the permutation

$$E = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 4 & 2 & 3 & 1 & 0 & 9 & 7 & 8 & 6 & 5 & 14 & 12 & 13 & 11 & 10 \end{bmatrix} \quad (5)$$

can be written as the product of three non-overlapping transformations,

$$E = E_1 \cdot E_2 \cdot E_3 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 & 8 & 9 \\ 9 & 7 & 8 & 6 & 5 \end{bmatrix} \cdot \begin{bmatrix} 10 & 11 & 12 & 13 & 14 \\ 14 & 12 & 13 & 11 & 10 \end{bmatrix} \cdot \quad (6)$$

The bands associated with each permutation E_j are the intervals

$$b(E_1) = [0..4]$$

$$b(E_2) = [5..9] \quad (7)$$

$$b(E_3) = [10..14]$$

The base values c_j of each band is just

$$c_1 = 0 \,;\, c_2 = 5 \,;\, c_3 = 10 \quad (8)$$

The transformation E can then be seen to obey equation (4) which in this case is written:

$$i + c_j \rightarrow e_{i,j} + c_j, \quad (9)$$

where $e_{1,0}$=4, $e_{1,1}$=2, $e_{1,2}$=3, $e_{1,3}$=1, $e_{1,4}$=0.

In accordance with a preferred embodiment of the present invention and as mentioned hereinabove, the scanner acquisition unit **112** receives a pixel value I_i from location i of the digitized image **120** and stores the pixel value in permuted location $z_i = p_i + q_b$ of memory **114**, where q_b is the starting memory location for band b. The permutation is typically performed according to a transformation P_j by an address generating routine $G(P_j)$, described in more detail hereinbelow. Upon termination of the permutation of the entirety of one band **122**, the portion of the permuted band and any previously unwritten portions of previous bands still stored in memory **114** which comprise an integral multiple of blocks of data is written to storage medium **118**. The remaining portion of the permuted band is typically written to storage medium **118** only upon termination of the permutation of the next band **122**. It is a feature of the present invention that the system of Fig. 1 writes a block of data to the storage medium **118** typically only when the block of data is full. It is an additional feature of the present invention that the permutation operation does not significantly slow down the writing to the storage medium versus writing in a unpermuted manner.

The acquisition and permutation of a second band **122** of pixel values can be performed in parallel with the write operation for a first already permuted band. This parallel reading and writing is known as double buffering.

For a double buffering scheme which neglects block size, of storage medium **118**, memory **114** is typically divided into two buffers. Each buffer is minimally the size of one permutation P_j. The size of a permutation is defined as the number of elements in a band.

For storage medium **118** organized in blocks, typically of a predetermined size B, the minimum size of memory **114** for a double buffering scheme is typically twice the size of the

permutations P_j plus twice the size B of one block. The addition of $2B$ is necessary because the bands may not be block aligned.

It will be appreciated that the maximum data buffer size is principally dependent on the size of the band **122** and secondarily dependent on both the block size, B, and the anticipated location of the band on storage medium **118**. It is not dependent on the complexity of the transformation.

As mentioned hereinabove, an address generating routine $G(P)$ is defined which generates the permuted location p_i in memory **114** for each pixel as it arrives sequentially. The address generating routine $G(P)$ is less general than the permutation P in that the address generating routine $G(P)$ is a routine which produces the permuted addresses sequentially, on successive routine executions, whereas the permutation P is a transformation which will produce any permuted address given its address as an argument.

According to a preferred embodiment of the present invention, the definition of the permutation P, and accordingly of permutation P_1 is typically chosen to reduce the number of i/o operations necessary to perform common processing operations on the digitized image **120**. For example, the resolution of a typical CRT monitor is 75 pixels per inch whereas the typical resolution of a digitized 8 1/2 in. by 11 in. image is 300 pixels per inch. Thus, typically only 1/16 of a stored image **120** can be displayed at one time on an 8 1/2 in. by 11 in. CRT monitor.

In order to display the image **120**, it is typically abstracted by a reduction factor r, that is, the displayed image includes every r th pixel of every r th row. For the abovementioned example, the reduction factor r is set to 4.

According to a preferred embodiment of the invention, the permutation P is typically defined such that the full-size image is stored as r^2 smaller images, known as abstracted images, that each have $1/r$ the number of both rows and columns. The full-size image is stored in the band structure described hereinabove where every band is organized into r^2 abstracted bands and where each abstracted band is written to the storage medium **118** as a continuous whole. An abstracted image can thus be read by reading only those blocks belonging to the abstracted bands comprising the abstracted image. It will be appreciated that the abovementioned embodiment presents a savings of generally r^2, neglecting disk granularity, over reading the entirety of the image and subsequently abstracting it.

Each abstracted image comprises every r th pixel of every r th row of the full-size image, but the initial pixel of each abstracted image differs among abstracted images. If the leftmost and topmost pixel is defined two-dimensionally as pixel (0,0) where the location numbers increase to the right and down, the initial pixels are chosen from the pixels belonging to the square whose corners are the pixels (0,0), (0,$r-1$), ($r-1$,0) and ($r-1$,$r-1$). In other words, the abstractions differ by a two- dimensional phase.

As mentioned hereinabove, each band **122** of the digitized image **120** is typically divided into r^2 abstracted bands where an abstracted band comprises only those elements of the band **122** belonging to one of the r^2 abstractions. Moreover, the number of rows and columns of the band **122** are typically multiples nr and mr, respectively, of r. The size of the abstracted band is mn and the size of band **122** is r^2 times the size of the abstracted band, or mnr^2. If enough memory **114** exists to store the entire image, n can be chosen so the entire image will occupy one band.

For each band, the incoming pixels belonging to a single abstracted band are located, in order, in a single continuous area of memory **114**.

Equations **(10a-e)** present the permutation equation P_1 of the first band **122** of image **120**.

$$\theta_c = i \bmod r \tag{10a}$$

$$\theta_r = \left\lfloor \frac{i}{rm} \right\rfloor \bmod r \tag{10b}$$

$$\lambda_c = \left\lfloor \frac{i \bmod rm}{r} \right\rfloor \tag{10c}$$

$$\lambda_r = \frac{i}{mr^2} \tag{10d}$$

$$p_i = mn(\theta_c + r\theta_r) + \lambda_c + m\lambda_r \tag{10e}$$

Where $\lfloor \cdots \rfloor$ is the floor function. In equations **(11a-e)** θ_c is the column geometric phase of pixel i, θ_r is the row geometric phase of pixel i, λ_c is the column within the abstracted band of pixel i, λ_r is the row within the abstracted band of pixel i. Equations **(11a-e)** comprise the inversion formula for converting permuted indices into sequential indices.

$$\theta_c = \left\lfloor \frac{p_i \bmod rmn}{nm} \right\rfloor \tag{11a}$$

$$\theta_r = \left\lfloor \frac{p_i}{rmn} \right\rfloor \tag{11b}$$

$$\lambda_c = p_i \bmod m \tag{11c}$$

$$\lambda_r = \left\lfloor \frac{p_i \bmod mn}{m} \right\rfloor \tag{11d}$$

$$i = \theta_c + mr\theta_r + r\lambda_c + r^2 m\lambda_r \tag{11e}$$

Software for the address generating routine, the permutation formula, and the inversion formula, written in the C++ language, are found in Annex A. The address generating routine is called `next` and the routine that produces the permutation function is called `q`. The address inversion generating routine is called `inext` and the inversion permutation function is presented in `qinv`. A test program, called `gatest`, to check the the equality of the address generating routines and the address permutation functions, and code to check that the address inversion permutation function correctly invert the address permutation function, are also included in Annex A.

Note that, although the address generating routine is complicated, it requires no multiplication operations, other than of constants which can be precomputed Thus, the address generating routine is typically fast enough to permute the address for a first pixel before scanner **110** acquires a second pixel.

Fig. 2A illustrates a sequentially received image and Fig. 2B illustrates the abovedescribed permutation definition for some abstractions of one 12x16 band of an image 130 abstracted by 4. The number of rows nr in a single band is 12, where n is 3, and the number of columns mr is 16, where m is 4. Thus, the size mn of an abstracted band is 12 pixels.

The band 131 will be abstracted into 16 abstracted bands where each of the 16 abstracted bands is referred to by two indices x,y representing its initial pixel, or geometric phase. If each abstracted image is denoted by A, then A(0,0) is the abstracted image which includes the pixels (0,0), (0,4), (4,0), etc..

Fig. 2B shows some of the 16 abstracted bands. Five of the abstracted bands are labelled according to the abstracted images A(x,y) to which they belong and the pixels stored in them are indicated. In the interests of clarity, the remaining abstracted bands are not filled in nor are they labelled. Fig. 2B illustrates the storage order of the abstracted bands on the storage medium 118. Fig 2C illustrates an alternative view of the abstracted bands shown in Fig. 2B wherein each abstracted band is shown as a rectangle corresponding to the original full scale image shown in Fig. 2A.

As mentioned hereinabove, the pixels belonging to an abstracted band are stored in a continuous area of memory 114. If the pixel 0, the first pixel of the abstracted band belonging to A(0,0), is stored in memory location 0 labelled 0, then the second pixel of the abstracted band of A(0,0), pixel 4, is stored in memory location 1 labelled 1. The memory locations for some indicative pixels are marked on Fig. 2B. If pixels are stored in more than 1 memory location then the memory locations indicated is understood to be multiplied by the number of memory locations needed to store a pixel in actual memory address calculations.

The first pixel of the abstracted band of A(1,0), pixel 1, is typically stored in the first location after the end of the abstracted band belonging to A(0,0). Since the size of each abstracted band is 12, the first location of the abstracted band belonging to A(1,0) is location 12. Accordingly, the abstracted band of A(2,0) begins in location 24, and the abstracted band belonging to A(0,1) begins in location 48.

As mentioned hereinabove, abstracted images A(x,y) each comprise $1/r^2$ of the pixels in image 130 but differ by a two-dimensional phase. It can be seen from Fig. 2B that the pixels of abstracted band A(0,0) come from the first, fifth, ninth and thirteenth columns of the first, fifth and ninth rows of band 130 whereas the pixels of abstracted band A(3,2) come from the fourth, eighth, twelfth and sixteenth columns of the third, seventh and twelfth rows of band 130.

For the example in Fig. 2B, for the pixel 73, i=73. Equations (10a-e) will give the results θ_c=1, θ_r=0, λ_c=2, λ_r=1 and p_{73}=18. Pixel 73 is in fact in the abstraction having the phase (1,0) and within its abstracted it is in column 2, row 1 (see Fig. 3C). Pixel 73 is in the 18th location sequentially from the first stored pixel stored at location 0. Using the inverse equations (11a-e) with p_i=18 we obtain θ_c=1, θ_r=0, λ_c=2, λ_r=1 and i=73.

It will be appreciated that if image 130 comprised more than one band, then the pixels belonging to the other bands would be stored in memory 114 in a manner similar to that described hereinabove. If double buffering is used, the pixels belonging to a second band are stored in the second buffer of memory 114 while the pixels already stored in the first buffer are being written to storage medium 118. For image 130 of more than one band, $A(x,y)$ comprises more than the

one abstracted band shown with respect to Fig. 2B.

An alternative embodiment of the invention defines the permutation P such that the digitized image is organized according to a multiplicity of reduction factors. For example, it may be desired to be able to abstract an image at a range of small reduction factors 2, 3, 4, 5, and 6, rather than the fixed factor 4 in the previous embodiment. To do so, a large reduction factor is found which is the least common multiple of the entirety of small reduction factors. For the example given, the large reduction factor might be 60 since each of 2, 3, 4, 5, and 6 are factors of 60. If it is desired to store the image according to the previous embodiment, the image would have to be stored as 3600 abstracted images. However the abstracted images of the different small reduction factors are not disjoint with respect to each other. Thus, an alternative embodiment of the present invention discloses a permutation P which facilitates access to the multiplicity of abstracted images at the small reduction factors by utilizing regions of overlap between them.

The following discussion uses an example of a large reduction factor of 60 but should not be construed to limit the invention to that example reduction factor.

To design the permutation P, the abstracted images at the smaller reduction factors are calculated from the 3600 possible abstracted images. An abstraction set S_r is defined such that its elements are pixels belonging to an abstracted image at reduction factor r which begins at the (0,0) pixel.

The intersections of the five abstraction sets S_2, S_3, S_4, S_5 and S_6 are shown in a Venn diagram in Fig. 3. A multiplicity of regions or subsets, denoted $S_a, \cdots S_k$ on Fig. 3, are shown. Subsets S_a, S_b, S_c, S_d, S_e, S_f, S_g and S_h are regions of overlap and belong to a multiplicity of sets; the remaining subsets, S_i, S_j and S_k, are independent regions which belong to the sets S_3, S_5 and S_2, respectively. Subsets $S_a, \cdots, S_k$ are defined as follows:

$$
\begin{aligned}
S_a &= S_3 \cap S_4 \cap S_5 \\
S_b &= (S_3 \cap S_4) - S_5 \\
S_c &= (S_5 \cap S_4) - S_3 \\
S_d &= (S_4 - S_5) - S_3 \\
S_e &= ((S_2 - S_4) \cap S_5) - S_3 \\
S_f &= ((S_2 - S_4) \cap S_3) - S_5 \\
S_g &= ((S_3 \cap S_5) \cap S_2) - S_4 \\
S_h &= (S_3 \cap S_5) - S_2 \\
S_i &= (S_3 - S_2) - S_5 \\
S_j &= (S_5 - S_2) - S_3
\end{aligned}
\qquad (12)
$$

$$S_k = ((S_2 - S_4) - S_3) - S_5$$

$$S_l = S_1 - S_2 - S_3 - S_5$$

For example: the subset that is common to all sets is S_a which is defined by equation 13,

$$S_a = \{A(0,0)\} \tag{13}$$

The subset S_c is defined in equation 14,

$$S_c = \{A(0,20), A(0,40), A(20,0), A(20,20), A(20,40), A(40,0), A(40,20), A(40,40)\} \tag{14}$$

As can be seen from equations 13 and 14, the size of a subset is typically the number of abstracted images $A(x,y)$ at the large reduction factor which belong to it. It will be appreciated that the subsets $S_a, \cdots, S_k$ are not of equal size, nor do they as a whole include the entirety of pixels of the image 120. Table 1 presents the sizes of the subsets.

The abstraction sets S_2, S_3, S_4, S_5 and S_6 can be written in terms of the subsets of Fig. 3 as follows:

$$S_2 = S_a \cup S_b \cup S_c \cup S_d \cup S_e \cup S_f \cup S_g \cup S_k$$

$$S_3 = S_a \cup S_b \cup S_f \cup S_g \cup S_h \cup S_i$$

$$S_4 = S_a \cup S_b \cup S_c \cup S_d$$

$$S_5 = S_a \cup S_c \cup S_e \cup S_g \cup S_h \cup S_j \tag{15}$$

$$S_6 = S_a \cup S_b \cup S_f \cup S_g$$

$$S_{10} = S_a \cup S_c \cup S_e \cup S_g$$

Table 1: Sizes of Subsets

Subset	Size	Subset	Size
S_a	1	S_g	3
S_b	24	S_h	12
S_c	8	S_i	288
S_d	192	S_j	96
S_e	24	S_k	576
S_f	72	S_l	2304

The subsets $S_a, \cdots, S_k$ enable the Venn diagram to be be ordered in a linear fashion to allow fast access to specific abstraction sets. A variety of permutations utilizing the subsets $S_a, \cdots, S_k$ can be performed; the optimal one will depend on the expected frequency of access to the various abstraction sets. One such ordering ensures that abstraction sets at smaller reduction factors which have more elements to them be less fragmented than the ones at larger reduction factors.

According to a preferred embodiment of the present invention, the multiplicity of abstraction sets, $S_2, \cdots, S_6$ in the example, are stored in the band structure described hereinabove. The band size is typically at least mnr^2 where r is the least common multiple used to create the abstracted images $A(x,y)$.

Fig. 4A illustrates an example permutation of a band **122** which typically enables abstracted bands of the example abstraction sets S_2, S_3, S_4 and S_6 to be read sequentially. In Figs 4B-4F the bold lines specify the subsets to be read to create abstracted sets $S_2, \cdots, S_6$ respectively. Abstraction set S_5 typically cannot be read sequentially. An alternative embodiment of the invention, shown in Fig. 5, enables abstracted bands of abstraction set S_5 to be read in one sequential read. A subset S_q is created by replacing the pixels in the set $S_q = S_c \cup S_a \cup S_f \cup S_g$ with pixels from the set S_l which are located close to the pixels of set S_q. For example, the approximation to the abstracted image A(0,0) is the abstracted image A(1,0) which is in the subset S_l.

It will be appreciated that Figs. 4 and 5 are not to scale.

The entirety of the image **120** is stored in bands **122**. Thus, the abstracted band belonging to subset S_l for each band **122** is typically stored in an area of both a buffer of memory **114** and of the storage medium **118** after the abstracted bands of subsets $S_a, \cdots, S_k$. Alternatively, the abstracted band of subset S_l can be used to block align the abstracted bands of subsets $S_a, \cdots, S_k$. In the alternative embodiment, elements in one abstracted band of subset S_l are located before and after the elements of the abstracted band subsets $S_a, \cdots, S_k$, located according to either of Figs. 4 or 5, such that at least one frequently accessed abstracted band begins at the beginning of a storage medium block.

The image typically contains a multiple of r rows and r columns. An image that does not conform to these constraints typically is expanded to conform.

For an image which is 8.5 inches wide, scanned at 300 pixels per inch where each pixel is 32 bits, the minimal number of rows for the example given herein needed to make a band is 60 rows. The resultant band typically occupies 597K bytes. The maximal expansion of the image due to the abovementioned constraint is 1/5 inch (60 pixels) in height and width and the average expansion is half this amount.

In accordance with the alternative embodiment of the invention, an abstracted image with reduction factor n, where n is a product of at least two of the factors r, can be read by reading the subsets which are common to the factors which make up n. Thus, the abstracted image at reduction factor 10 comprises the subsets S_a, S_c, S_e, and S_g.

If n is a multiple of one of the factors r, its abstracted image can be read by reading the abstraction set S_r of which n is an integral multiple and further abstracting S_r. For example, the abstracted image with reduction factor 9 is read by reading the abstracted image S_3 and further

abstracting it by a factor of 3.

An abstracted image with a reduction factor of n, where n is relatively prime with reference to the factors r or is not an integral scale factor, can be obtained by reading an abstracted image with a smaller reduction factor and the resampling the lower order abstracted image to obtain the desired abstracted image. For example, the abstracted image with a reduction factor of 7 can be obtained by reading the abstracted image S_4 interpolating by 7/4.

It will be appreciated that the abovedescribed subset ordering and its inverse can be precomputed and stored in the scanner acquisition unit **112** in a lookup table. The lookup table will enable a fast calculation of the locations of the permuted pixels. Annex B contains a program to generate the permutation and its inverse for the multiple abstracted image permutation described hereinabove. This program makes use of the base class GABS defined in Annex A.

Another example of a multiplicity of reduction sizes that is possible with the abovedescribed embodiment is fixed power reductions. Typically a set of reductions that are successive powers of 2 are needed, that is multiple reductions of order 2,4,8,16,32,... For example, if r is 64 the 2,4,8,16,32,64 order reductions can be read from the storage medium in one access. Reduction factors of powers of 2 are useful in the pyramid transformation. The pyramid transformation and its use is described in detail in the book, A. Rosenfeld, ed. *Multiresolution Image Processing and Analysis.* Springer-Verlag, Berlin, 1984. It should be noted that, in general, each pixel in the reduced order image of the pyramid transformation may be a complex mathematical transformation of the full resolution image data. The abovedescribed embodiment derives the reduced image pixels from a simple subsampling of the original image.

The permutation equations for the abovedescribed transformation are slightly different than the equations **(10e)** and **(11e)**. Since the r^2 abstracted bands are not stored in a simple order dictated by the geometric phase, a mapping $M(\theta_r,\theta_c)$ of each abstracted band is needed in order to compute the transformation. This mapping returns the sequential index of the location of the abstracted band $A(\theta_r,\theta_c)$ within the subset ordering as illustrated in Fig. **4A**. Equation **(10e)** is then replaced by equation **(16)** for this transformation.

$$p_l = mnM^{-1}(\theta_c,\theta_r) + \lambda_c + m\lambda_r \tag{16}$$

Equation **(11e)** for the inverse is replaced by three new equations **(17a-c)**.

$$\gamma_c = M(\theta_c,\theta_r) \bmod r \tag{17a}$$

$$\gamma_r = \left\lfloor \frac{M(\theta_c,\theta_r)}{r} \right\rfloor \tag{17b}$$

$$i = \gamma_c + mr\gamma_r + r\lambda_c + r^2 m\lambda_r \tag{17c}$$

If the digitized image **120** is stored in a format defined by either of the permutations disclosed hereinabove, or in any other non-sequential format, it typically can be transformed to another, more common, format for additional processing. A typical processing format is the sequential format. For instance, the image **120** may be stored as abstracted images of reduction factor 4 and it may be desired to perform a convolution on the entirety of the image **120**. The sequential format is the natural format for this calculation. Thus, the stored abstracted image is

typically transformed into the sequential format. This can be accomplished by simply transforming the data read from the storage medium **118** via the inverse of the transformation by which it was stored, as shown in equation **18**.

$$P^{-1} = P_1^{-1} \cdot P_2^{-1} \cdots P_b^{-1} \tag{18}$$

P^{-1} is typically performed on a first band of data in parallel with the reading of a subsequent band of stored data from the storage medium **118**. Since the time to transform the data is typically less than the time to read the stored data from the storage medium **118**, the conversion from one format to the other generally does not take longer than reading the stored data without any format conversion. The size of memory **114** needed to transform the data for a double buffering scheme is bounded by twice the size of the largest band of the transformed image.

According to another embodiment of the invention, an image can be transformed from one band structure format F_1 to another band structure format F_2. The method comprises the following steps:

a. Read as many bands of stored image data stored in the F_1 format as necessary to produce enough data to fill a band in format F_2.

b. Convert the data in format F_1 to sequential format.

c. Convert the data in sequential format to format F_2. Convert only as much data as necessary to produce an integral number of bands of format F_2.

d. Store the integral number of bands of F_2 on storage medium **118**.

e. Free the memory **114** of the integral number of bands in format F_2.

f. Return to step a. and continue until no more data is available.

Fig. **6** illustrates an example of the method of transforming between two formats as described hereinabove. An image **150** is stored in format F_1 whose band structure comprises equal bands of 100K pixels each. An image **152** is stored in format F_2 whose band structure comprises equal bands of 150K pixels each. To transform image **150** to image **152**, the following steps are typically followed.

1. Read the first two bands B_a and B_b of image **150** and transform them into sequential format.
2. Transform the entirety of band B_a and half of band B_b into band B_1 from sequential format to format F_2.
3. Write band B_1 to image **152**.
4. Free the space in memory **114** currently occupied by band B_a and by the first half of band B_b.
5. Read in band B_c of image **150** and convert to sequential format.
6. Transform the second half of band B_b and the entirety of band B_c into band B_2 from sequential format to format F_2.
7. Write band B_2 to image **152**.
8. Free up the space in memory **114** currently occupied by the second half of band B_b and by the entirety of band B_a.

9. Repeat until no more data remains in image 150.

The maximum space in memory 114 required for the method of converting between two formats described hereinabove, when using a double buffering scheme, is typically twice the sum of the band size required for each of the formats. It is a feature of the invention that the pixels of image 150 are read only once.

If the band structure of the two formats F_1 and F_2 are equivalent, the data can be easily transformed without first transforming the data into the sequential format. The permutation is $P_{F_2}^{-1} \cdot P_{F_1}$, where $P_{F_2}^{-1}$ is the transformation from format F_2 to sequential format and P_{F_1} is the transformation from sequential format to format F_1. Alternatively, if the transformation between the two formats F_1 and F_2 is frequent, the permutation can be precomputed, thus reducing the complexity of the transformation and the time needed to execute the transformation. The maximum space in memory 114 required for this alternative method when using a double buffering scheme is typically twice the size of the band.

According to an embodiment of the present invention, the image data can be rotated at an arbitrary angle using a tile transformation which is well known in the art. The tile transformation is described in the Pixar™ manual, *Scope, Release 1.1, Genera 7.1* by The Graphics Division of Symbolics Inc., p. TU 2, which is incorporated herein by reference. The tile transformation consecutively stores pixels which are elements of a square section of the image in a single block on the storage medium 118. A tile of size u^2 comprises u rows of pixels each having u columns. Typically, a tile will occupy a block. For a block size of 4096 bytes and pixels of 4 bytes each, a 32x32 tile is typically chosen.

Fig. 7 illustrates the transformation for sequential format image 130 of Fig. 2A. Shown are 5 of the 12 4x4 tiles in which the first 192 pixels are stored. The first tile 160, comprising the first four pixels of the first four rows, is stored consecutively in an area of memory 114. The second tile 162, comprising the second four pixels of the first four rows, is stored after the end of the first tile 160. Thus, if first tile 160 begins at memory location 0, then second tile 162 begins at memory location 16. If pixels are stored in more than 1 memory location then the memory locations indicated is understood to be multiplied by the number of memory locations needed to store a pixel in actual memory address calculations.

The permutation P for the abovementioned example is given in equation 19.

$$p_i = i \bmod u + u^2 \left(\left\lfloor \frac{1}{u} \right\rfloor \bmod m + u \left\lfloor \frac{i}{um} \right\rfloor \right) \tag{19}$$

Where m is the number of tiles in a row of the image.

To rotate, by 90°, the image which is stored in tile format a block size tile is read into a buffer of memory 114, moved to the block it will occupy when it is rotated, and then rotated. The operation is shown in Fig. 8. The operation can alternatively be performed in one read/write pass of the image by reading a tile is read into a buffer of memory 114 and directly writing it to the corresponding destination rotated by 90°. This alternative method typically requires enough space in memory 114 to hold two tiles, the tile read in and the rotated tile. If double buffering is desired, to increase the speed of the computation, four buffers are typically allocated.

According to a preferred embodiment of the invention, rotation at an arbitrary angle using a tile transformation can be performed. Rotation by 45^0, being the rotation which typically uses the most buffer space in memory **114**, is illustrated in Fig. **9** for an image organized into a 5x5 tile format.

The fact that the resultant rotated image is non-rectangular is irrelevant to the current discussion. Well know techniques can be used to convert the rotated image to a rectangular form. For example, the rotated image can either be clipped to rectangular form, or expanded to contain the rotated image.

For the example in Fig **9**, the rotation operation takes a 5x5 image **172**, shown rotated in Fig. **9**, rotates it and writes it to storage **118** as a 5x5 image **174**. The tiles of source image **172** are denoted with the double capital letters *AA–YY*, those of destination image **174** are denoted with the double small letters *aa–yy*. Image **172** is organized into bands **176**, **178**, **180**, **182** and **184** where each band comprises five tiles.

Typically a source image **172** tile affects number of destination image **174** tiles. This dependency is denoted by $\delta=D(\sigma)$, where δ is a set of destination image **174** tiles and σ is a set of source image **172** tiles. For example, $D(\{AA\})=\{cc\}$ and $D(\{BB\})=\{cc,dd,ii\}$. The function D is dependent on the details of the calculation of each destination pixel since each destination pixel typically is dependent on a multiplicity of source image pixels.

According to a preferred embodiment of the rotation operation tiles of source image **172** are buffered as follows.

1. Read into memory **114** the first two rows of tiles from the source image **172**.
2. Read into memory **114** the next row of tiles from source image **172**.
3. Calculate and write to destination image **174** all destination tiles that only dependent on the current three rows of source image **172** tile.
4. Free the memory occupied by the last recently read row of source image **172** tiles.
5. Repeat steps 2-4 until the entirety of the image has been written.

An example of the abovementioned embodiment reads from the storage medium **118** bands of tiles labelled **176**, **178**, **180** from image **172** and writes out all rotated tiles that are fully defined by the tiles of image **172** currently in memory **114**, being tiles *ee*, *dd*, *jj*, *cc*, *ii*, *oo*, *bb*, *hh*, *nn*, and *tt*. The buffers for the first band of tiles **176** can then be freed and the band **182** read from the storage medium **118** into the location previously occupied by band **176**. All previously unwritten rotated tiles that are fully defined by the unrotated tiles currently in memory **114**, being tiles *aa*, *gg*, *mm*, *ss* and *yy*, are written to storage medium **118**. The last recently unrotated band **178** is freed and the next unrotated band **184** is read in from storage medium **118**. All previously unwritten rotated tiles that are fully defined by the unrotated tiles currently in memory **114**, being tiles *ff*, *ll*, *rr*, *xx*, *kk*, *qq*, *ww*, *pp*, *vv*, and *uu*, are written to storage medium **118**.

This procedure is summarized in Table 2.

It should be appreciated that the only 3 bands of unrotated tiles need be stored in memory at any one time for any size image that is to be rotated and for any angle of rotation. This is because any destination tile can only be dependent on source tiles which span three sequential rows of tiles.

Table 2.

Read and buffer tiles	Discard tiles	Write tiles
AA,BB,CC,DD,EE FF,GG,HH,II,JJ KK,LL,MM,NN,OO		
		ee,dd,jj,cc,ii,oo bb,hh,nn,ll
	AA,BB,CC,DD,EE	
PP,QQ,RR,SS,TT		
		aa,gg,mm,ss,yy
	FF,GG,HH,II,JJ	
UU,VV,WW,XX,YY		
		ff,ll,rr,xx,kk qq,ww,pp,vv,uu

The total memory **114** requirements for this method is 3 bands of tiles, to buffer the unrotated tiles, plus one tile to hold the rotated tile. If double buffering is to be used the total memory **114** requirements is 4 bands of tiles to buffer the unrotated tiles, plus 2 tiles for the space to store rotated tiles before they are written to storage medium **118**.

An alternative embodiment of the rotation operation utilizes virtual memory to store the entirety of the rotated image **174**. A Least Recently Used (LRU) page replacement policy causes the last three bands to remain resident in memory **114**. If there are at least enough pages available to store 4 bands concurrently, on the order of $O(f)$ page faults typically occur, where f is the number of pages in the rotated image **172**. Upon termination of the rotation of rotated image **172**, the entirety of the rotated image **172** is written to storage medium **118**. Alternatively, if the operating system of CPU **116** supports direct memory mapped disk files, the rotated image can be directly mapped to the virtual memory of CPU **116** and the rotated image written to storage medium **118** via standard virtual memory mechanisms. This alternative embodiment saves system swap area and the final writing of the image from virtual memory to the storage medium **118**.

According to an embodiment of the present invention, images stored in any band format can be rotated. The steps 1-6 are as follows.

1. Read in and convert to tile format a sufficient number of bands **122** of the image so that after conversion to tile format at least two bands of tiles are in memory.

2. Read in and convert to tile format a sufficient number of bands **122** of the image so that after conversion an additional band of tiles is present in memory
3. Calculate the entirety of rotated tiles that are only dependent on the converted three bands of unrotated tiles. Write these tiles to a disk file in tile format.
4. Free the storage for the least recently read band of unrotated tiles.
5. Repeat steps 2, 3 and 4 until the entirety of image **120** has been read in and converted to tile format.
6. Use the method mentioned hereinabove to convert the rotated image currently stored on disk in tile format to a rotated image in the original band format.

It should be noted that in the abovementioned rotation methods the image **120** is rotated by reading and writing the entirety of two image-sized files.

Reference is now made back to Fig. **1**. The scanner **110** is connected to the scanner acquisition unit **112** via a parallel interface, such as a Digital Equipment Corporation (DEC) DR11 interface defined by DEC and documented in the reference *Microcomputer Interface Handbook*, Digital Equipment Corporation, 1980. The CPU **116**, memory **114** and scanner acquisition unit **112** are connected together on a CPU data and address buses **202**. A similar CPU bus structure and DMA scheme is described in Intel technical manual *82380 High Performance 32-bit DMA Controller with Integrated System Support Peripherals*, Intel Corp 1988. As mentioned hereinabove, the system illustrated in Fig. **1** is a variant of standard DMA architectures.

The system operates as follows: A *NEW_DATA_READY* line of the scanner **110** strobes, indicating to the scanner acquisition unit **112** that a valid pixel value I_i (or a color separated pixel value representing one color of a color separated pixel) is available on the *DATA* lines of the scanner **110**. Since there is typically no handshake between the scanner **110** and the scanner acquisition unit **112**, the acquisition unit **112** must be able to receive the pixel value at a rate greater than or equal to the rate that the scanner **110** acquires the pixel value.

Upon pulsation of the *NEW_DATA_READY* line, the *HOLD* control line typically is asserted, thus requesting control of the data and address buses **202** from the CPU **116**, or other permanent bus master.

When CPU **116** relinquishes control of the data and the address buses **202**, it asserts the *HLDA* control line and enters a hold-state until the scanner acquisition unit **112** drives the *HOLD* signal false.

With the *HLDA* signal asserted, the scanner acquisition unit **112** places an address on the address bus and the valid pixel value on the data bus **202**. The address is the permuted memory location z_i for the pixel value and is calculated according to any of the methods of the present invention. The scanner acquisition unit **112** generates a memory write signal and the pixel value I_i is written to the proper address in memory **114**.

According to a preferred embodiment of the present invention, the permutations described hereinabove are implemented as address generator routines typically stored in the scanner acquisition unit **112** as microcode programs. The programs in Annex A and B are typically compiled into microcode to implement an address generator routine $G(P)$.

In an alternative embodiment of the present invention the permutations are generated by precomputed lookup tables stored in ROM (read only memory) or lookup tables computed just prior to the image acquisition. The lookup tables are computed by the address generator routine $G(P)$.

It will be appreciated that the address generator routine $G(P)$ of the present invention is significantly different from those of standard DMA controllers which merely generate a new address by incrementing the previous address by a predetermined value.

Once the pixel value is written to its location in memory **114**, the scanner acquisition unit **112** deasserts the *HOLD* signal.

Subsequently, the CPU **116** deasserts the *HLDA* signal and continues processing. The above described process is repeated for each new pixel value I_i. After a band **122** has been completely written into a first buffer of memory **114**, the scanner acquisition unit **112** signals this fact to the CPU **116** via the EOP signal which typically causes a CPU **116** interrupt. The CPU **116** then instructs the scanner acquisition unit **112**, by writing to i/o or memory mapped registers on the scanner acquisition unit **112**, to switch the base address to a second buffer of memory **114**, containing the location of the next buffer. The band of data stored in the first buffer is transferred to storage medium **118** via the storage medium DMA controller **119**, which may typically contain an 82380 DMA controller from Intel corporation. The storage medium DMA controller **119** and the storage medium **118** are typically chosen to transfer the band of data at a faster rate than that by which the scanner **110** acquires image data. Thus, the first buffer will be emptied before the second buffer is completely filled.

It should be noted that this technique must be slightly modified for systems with more more than one DMA unit on the system bus. A bus arbitration protocol (for example, daisy chaining or priority resolution) must be instituted for the *HOLD* signal. These well known techniques are described in *Microprocessor System Design Concepts*, Nikitas A, Alexandridis, Computer Science Press, 1984, specifically sections 9.7.3 and 9.8.3.

It will be appreciated by persons skilled in the art that the present invention is not limited by what has been particularly shown and described hereinabove. The scope of the present invention is defined only by the claims which follow:

ANNEX A

The following file named gabs.cc defines a base class GABS that is used in both Annex A and Annex B.

```
1  //  This file is called gabs.cc
2  //  It is writen in the C++ language as described in the book:
3  //  "The C++ programming Language" by Bjarne Stroustrup,
4  //  1986, Addison-Wesley.

5  class GABS {        // The abstract class for reduction permutations
6  protected:
7    int last_ptr;              // The pointer to the begining of the last
8                               //      abstracted block.
9    int column;                // The column in the image.
10   int row_in_abs_image;      // A flag to indicate when to go back to
11                              //      the first abstraction.
12   int ad;                    // Addressing within an abstracted band.
13   int grow;                  // Flag to signify when to start a new
14                              //      row in the abstracted band, i.e.
15                              //      after r image rows.
16   int inv_ad;                // Addressing for inverse transformation
17   int inv_count;             // Counter to keep track of abstracted band
18   int inv_rowcount;          // Counter to keep track of which row we are in.
19   int inv_ptr;               // The index of the current abstracted band.
20   int n;                     // n, number of rows in each abstracted band.
21   int m;                     // m, number of columns in each abstracted band.
22   int r;                     // r, order of the abstraction.
23   int r2;                    // r^2
24   int b_size;                // nmr^2, size of band in pixels
25   int phase;                 // The phase column.
26   pixel** abs_bases;         // Lookup table for abstract block addresses.
27   pixel** iabs_bases;        // Lookup table for inverse abstract block addresses.
28 public:
29   GABS(int nn, int mm, int rr) // A constructor for the GABS class.
30   {
31     n= nn;
32     m= mm;
33     r= rr;
34     r2= r*r;
35     b_size= n*m*r2;
36     abs_bases= new pixel*[r2];   // The table size is r^2.
37     iabs_bases= new pixel*[r2]; // The table size is r^2.
38     init();
39   }
40   // This is the part of the initialization that is repeated
41   // For each band.
42   void init()
43   {
```

```
44        inv_ad= -r;
45        inv_count= 0;
46        inv_rowcount= 0;
47        inv_ptr= 0;
48        last_ptr= -1;
49        column= -1;
50        phase= -1;
51        row_in_abs_image= 0;
52        ad= 0;
53        grow=0;
54      }
55
56      inline int band_size() // Return band size.
57      {
58        return(b_size);
59      }
60
61      // The next function is called to generate the next
62      // address when the next pixel value is presented
63      // by the scanner.
64      pixel* next() {
65        column++;                    // Increment the current column.
66        phase++;                     // Increment the current phase.
67        last_ptr++;                  // Increment to the next abstract block..
68        if(phase >= r)               // The variance of phase within a row is r.
69        {
70          ad++;
71          phase= 0;                  // Reset the phase.
72          last_ptr-= r;              // Jump back r abstracted blocks.
73        }
74        if(column >= (r*m))          // Are we at the end of a scanner row.
75        {
76          column= 0;                 // Reset column.
77          last_ptr+= r;              // Jump forward r abstracted blocks.
78          row_in_abs_image++;        // We are now in the next row of abstracted blocks.
79                                     // We have only nr rows of abstracted blocks.
80                                     //      After each r scanner row jump the next row.
81          if(row_in_abs_image >= r)
82          {
83            row_in_abs_image= 0;
84            last_ptr= 0;             // Start again at abstract block zero.
85          }
86          grow++;                    // Keep track of rows.
87                                     // If we have read r rows go.
88                                     // to next row in the abstract image.
89          if(grow >= r)
90          {
91            grow=0;                  // Go to the next row in the next abstracted band.
```

```
92          }
93          else
94          {
95            ad-= m;   // Continue on the same row in the next abstracted band.
96          }
97        }
98        // Return the sum of the abstract block pointer
99        //      and the addressing within the block.
100       return(abs_bases[last_ptr] + ad);
101     }

102     pixel* inext() {
103       inv_ad+= r;
104       if(inv_rowcount++ >= m)
105       {
106         inv_ad+= m*r*r - m*r;
107         inv_rowcount= 1;
108       }
109       if(inv_count++ >= m*n)
110       {
111         inv_ptr++;
112         inv_ad= 0;
113         inv_count= 1;
114       }

115       return (iabs_bases[inv_ptr] + inv_ad);
116     }

117     // qa  is the routine to compute the
118     // addressing within an abstracted band
119     int qa(int abs_col, int abs_row)
120     {
121       return(abs_col + abs_row*m); // Map pixels to sequential positions.
122     }

123     // qinvabs  is the routine to compute the
124     // inverse addressing within an abstracted band
125     int qainv(int abs_col, int abs_row)
126     {
127       return (r*abs_col + r2*m*abs_row);
128     }
129
130     // This routine computes the location of the abstracted band
131     // having the phase  (phase_row,phase_col)
132     // This routine is overridden in the actual class.
133     // It is an error to call the qabs routine from
134     // the class GABS.
135     virtual int qabs(int phase_col, int phase_row)
```

```
137     {
138       cerr << "Shouldn't get here";
139     }
140
141     // This routine computes the base location of first pixel of the abstracted
142     // band having the phase  (phase_row,phase_col).
143     // This routine is overridden in the actual class.
144     // It is an error to call the qinvabs routine from
145     // the class GABS.
146     virtual int qinvabs(int phase_col, int phase_row)
147     {
148       cerr << "Shouldn't get here";
149     }
150
151     // Compute the mapping to abstracted format given a scanner row index.
152     int q(int scanner_row_index)
153     {
154       // phase_col  is the column phase of the pixel.
155       // phase_row  is the row phase of the pixel.
156       // abs_col  is the column number within the abstracted band.
157       // abs_row  is the row number within the abstracted band.
158       int phase_col, phase_row, abs_col, abs_row;
159       phase_col= scanner_row_index%r;
160       phase_row= (scanner_row_index/(r*m))%r;
161       abs_col= scanner_row_index%(r*m)/r;
162       abs_row= scanner_row_index/(r2*m);
163       // The mapping is the sum of the location of the abstracted band  (qabs)
164       // and the location with that abstracted band  (qa).
165       return (m*n*qabs(phase_col, phase_row) + qa(abs_col, abs_row));
166     }
167
168     // Compute the mapping to scanner row index given a abstracted format index
169     int qinv(int abs_format_index)
170     {
171       int phase_col, phase_row, abs_col, abs_row;
172       phase_col= abs_format_index%(r*m*n)/(m*n);
173       phase_row= abs_format_index/(r*m*n);
174       abs_col= abs_format_index%m;
175       abs_row= (abs_format_index%(m*n))/m;
176       return (qinvabs(phase_col, phase_row) + qainv(abs_col, abs_row));
177     }
178   };
```

The following file named ga.cc defines a derived class of GABS called GA that implements the fixed size reduction embodiment.

```
1  // This file is called ga.cc
2  #include <stream.h>
3  #include "gabs.cc" // the file that defines class GABS


4  // Class GA is derived from class GABS it defines an ordering
5  // of the abstracted bands

6  class GA: public GABS {
7  public:
8    // Initialize an instance of class GA
9    // The array abs_bases is a table that gives the
10   // starting location of each abstracted band
11   // The array iabs_bases is a table that gives the
12   // upper left hand corner of the abstracted band in the original band
13   GA(pixel* offset, int nn, int mm, int rr) : GABS(nn,mm,rr)
14   {
15     for(int i=0;i<r2;i++)
16     {
17       abs_bases[i]= offset + i*m*n;
18       iabs_bases[i]= offset + i%r + r*m*(i/r);
19     }
20     // Note: the base address of the band in memory should be added to abs_bases.
21   }

22   // This defines the location of each abstracted band
23   int qabs(int phase_col, int phase_row)
24   {
25     return (phase_col + r*phase_row);
26   }
27

28   // This defines the location of the first pixel of the indicated abstracted band
29   int qinvabs(int phase_col, int phase_row)
30   {
31     return (phase_col + m*r*phase_row);
32   }
33 };
```

The following file named `gatest.cc` tests the class `GA`.

```
1  typedef char pixel;          // Use an example 32 bits per pixel
2  #include "ga.cc"

3  pixel* offset;               // Should point to the begining of the band buffer
4  main()
5  {
6    GA ga(offset, 3, 4, 4); //Create an instance of a mapping having n=3,m=3,r=4
7    int j, i;
8
9    // Check that the address generator and the mapping function return the
10   // same address. The multiplication by the size of a pixel is because
11   // the generator returns addresses while the mapping functions returns
12   // the permutation
13   for(i= 0; i < ga.band_size(); i++)
14   {
15     if(ga.next() != offset + ga.q(i))
16     {
17       cout << "Error in assertion: " << i;
18     }
19   }

20   // Reinitialize the instance of the mapping-generator
21   ga.init();
22   // Check that the inverse address generator and the inverse mapping function
23   // return the same address.
24   for(i= 0; i < ga.band_size(); i++)
25   {
26 //     cout << int(ga.inext()) << " " << ga.qinv(i) << "0;
27     if(ga.inext() != offset + ga.qinv(i))
28     {
29       cout << "Error in inverse assertion: " << i ;
30     }
31   }

32   // Reinitialize the instance of the mapping-generator
33   ga.init();
34   // Check that the mapping function and the inverse mapping function
35   // are in fact inverses of each other.
36   for(i=0;i < ga.band_size(); i++)
37   {
38     int j;
39     if(i != ga.qinv(ga.q(i))) // Check that i=q^-1(q(i))
40       cout << "Inversion error: " << i;
41   }
42 }
```

ANNEX B

The following file named gm.cc defines a derived class of GABS called GM that implements the multiple size reduction embodiment.

```
1  // This file is called gm.cc
2  #include <stream.h>
3  #include "gabs.cc" // the file that defines class GABS
4  #include "cluster.hh"
5  extern int mapping[];
6  static int* imapping;
7  // Class GM is derived from class GABS it defines an ordering
8  // of the abstracted bands
9  // This is the multiple order abstraction format.

10 class GM: public GABS {
11 public:
12   // Initialize an instance of class GM
13   // The array abs_bases is a table that gives the
14   // starting location of each abstracted band
15   // The array iabs_bases is a table that gives the
16   // upper left hand corner of the abstracted band in the original band
17   // We here take the specific case of r=60 and we execute the function
18   // ma() which defines the array mapping.
19   GM(pixel* offset, int nn, int mm) : GABS(nn,mm,60)
20   {
21     if(ca.starting_location == 0)
22       ma();
23     imapping= new int[60*60];  // The inverse mapping
24     for(int i=0;i<r2;i++)
25     {
26       abs_bases[mapping[i]]= offset + i*m*n;
27       iabs_bases[i]= offset + mapping[i]%r + r*m*(mapping[i]/r);
28       imapping[mapping[i]]= i; // Create inverse mapping array
29     }
30 }

31 // The function qabs and qinvabs are used in the generating functions.
32
33 // qabs defines the location of each abstracted band
34 int qabs(int phase_col, int phase_row)
35 {
36   return imapping[(phase_col + r*phase_row)];
37 }
38

39 // This defines the location of the first pixel of the indicated abstracted band
40 int qinvabs(int phase_col, int phase_row)
41 {
```

```
42        int mapped_abs_band= mapping[phase_col + r*phase_row];
43        int true_phase_col= mapped_abs_band % r;
44        int true_phase_row= mapped_abs_band / r;
45        return (true_phase_col + m*r*true_phase_row);
46      }
47    };
```

The following file named `set.hh` defines the basic operations on sets.

```
1   // This File contains support routines for the class AbsSet.
2   // It uses the BitSet class as described in the manual
3   // "User's Guide to the GNU C++ Library" by Doug Lea.
4   // Copyright 1988 Free Software Foundation..
5   // This library and the C++ compiler can be obtained from the
6   // Free Software Foundation, Inc., 675 Mass AVE, Cambridge, MA 02139.

7   #include <BitSet.h>
8   const int n= 60;
9   // the function rc2can converts row, column coordinates to
10  // canonical form.
11  inline int rc2can(int r, int c) { return r + n * c; }
12  class AbsSet
13  {
14    BitSet* bs;        // This provides the basic Set functionality.
15    int* ref_count;
16  public:
17    AbsSet();
18    AbsSet(int,int =0, int =0);
19    AbsSet::AbsSet(BitSet&);
20    ~AbsSet();
21    void operator = (AbsSet&);
22    AbsSet operator & (AbsSet& x);
23    AbsSet operator | (AbsSet& x);
24    AbsSet operator - (AbsSet& x);
25    void operator |= (AbsSet& x);
26    int operator == (AbsSet& x);
27    int operator != (AbsSet& x);
28    int operator <= (AbsSet& x);
29    void set(int r, int c) { bs->set(rc2can(r, c));}
30    int count() { return bs->count(1);}
31    cluster append(int*&);
32  };


33  AbsSet::AbsSet(int s, int r, int c)
34  {
35    bs= new BitSet;
36    ref_count= new int;
37    *ref_count= 1;
38    for(int i=0;i<60;i+=s)
39          for(int j=0;j<60;j+=s)
40            set(i+r, j+c);
41  }


42  AbsSet::AbsSet()
```

```
43  {
44    bs= new BitSet;
45    ref_count= new int;
46    *ref_count= 1;
47  }

48  AbsSet::AbsSet(BitSet& b)
49  {
50    bs= new BitSet(b);;
51    ref_count= new int;
52    *ref_count= 1;
53  }

54  AbsSet::~AbsSet()
55  {
56    if(*ref_count == 1)
57          delete bs;
58    else
59          (*ref_count)--;
60  }

61  void AbsSet::operator = (AbsSet& x)
62  {
63    if(*ref_count == 1)
64    {
65          delete bs;
66    }
67    else
68    {
69          *ref_count--;
70    }
71    (*x.ref_count)++;
72    ref_count= x.ref_count;
73    bs= x.bs;
74  }

75  AbsSet AbsSet::operator & (AbsSet& x)
76  {
77    return(AbsSet((*bs) & *(x.bs)));
78  }

79  AbsSet AbsSet::operator | (AbsSet& x)
80  {
81    return(AbsSet((*bs) | *(x.bs)));
82  }

83  AbsSet AbsSet::operator - (AbsSet& x)
84  {
```

```
85    return(AbsSet((*bs) - *(x.bs)));
86  }

87  int AbsSet::operator == (AbsSet& x)
88  {
89    return((*bs) == *(x.bs));
90  }


91  int AbsSet::operator != (AbsSet& x)
92  {
93    return((*bs) != *(x.bs));
94  }


95  int AbsSet::operator <= (AbsSet& x)
96  {
97    return((*bs) <= *(x.bs));
98  }

99  void AbsSet::operator |= (AbsSet& x)
100 {
101   *bs |= *x.bs;
102 }

103 cluster AbsSet::append(int*& ip)
104 {
105   cluster cl;
106   cl.size_of_set= 0;
107   cl.starting_location= cl.total_size_of_sets;
108   for(int i= bs->first(); i >= 0; i=bs->next(i))
109   {
110     *ip++= i;
111     cl.size_of_set++;
112   }
113   cl.total_size_of_sets+= cl.size_of_set;
114   return cl;
115 }
```

The following file named `cluster.hh` defines the data structure that allows the program to access the linear ordering of the sets $S_a, \cdots, S_k$.

```
1  // This defines the clustering of each set within the linear ordering of sets.
2  struct cluster
3  {
4    static total_size_of_sets;   // How big is the mapping so far.
5    int starting_location;       // Where does this cluster start.
6    int size_of_set;             // How big is the set.
7  };

8  extern cluster ca, cb, cc, cd, ce, cf, cg, ch, ci, cj, ck, csp1;
```

The following file named ma.cc calculates the subsets $S_a, \cdots, S_k$.

```
#include "cluster.hh"
#include "set.hh"
// This function creates the sets a–k. I also clusters the sets in the
// order detailed in the example.
int mapping[3600];
cluster ca, cb, cc, cd, ce, cf, cg, ch, ci, cj, ck, csp1;
ma()
{
  int *ip;

  // Create the set S_i.
  AbsSet s1(1);                          // S1
  AbsSet s2(2);                          // S2
  AbsSet s3(3);                          // S3
  AbsSet s4(4);                          // S4
  AbsSet s5(5);                          // S5
  AbsSet s6(6);                          // S6
  AbsSet sp1(((s1 - s2) - s3) - s5);     // S'1

  AbsSet a= s3 & s4 & s5;                // The set Sa
  AbsSet b= (s3 & s4) - s5;
  AbsSet c= (s5 & s4) - s3;
  AbsSet d= (s4 - s5) - s3;
  AbsSet e= ((s2 - s4) & s5) - s3;
  AbsSet f= ((s2 - s4) & s3) - s5;
  AbsSet g= ((s3 & s5) & s2) - s4;
  AbsSet h= (s3 & s5) - s2;
  AbsSet i= (s3 - s2) - s5;
  AbsSet j= (s5 - s2) - s3;
  AbsSet k= ((s2 - s4) - s3) - s5;

  // The order of the sets a–k are as outlined in Fig. 4
  ip=mapping;
  cj= j.append(ip);
  ce= e.append(ip);
  ck= k.append(ip);
  cd= d.append(ip);
  cc= c.append(ip);
  ca= a.append(ip);
  cb= b.append(ip);
  cf= f.append(ip);
  cg= g.append(ip);
  ch= h.append(ip);
  ci= i.append(ip);
  csp1= sp1.append(ip);
}
```

The following file named `gmtest.cc` tests the class `GM`.

```
 1  typedef int pixel;        // Use an example 32 bits per pixel
 2  #include "gm.cc"
 3  const int nn= 1;
 4  const int mm= 9;

 5  pixel* offset;            // Should point to the begining of the band buffer

 6  void is_scaled_image(pixel*, cluster**, int, int, int);

 7  main()
 8  {
 9    int j, i;
10
11    GM gm(offset, nn, mm); //Create an instance of a mapping having n=1,m=9,r=60
12  #ifdef NOCHECK
13    // Check that the address generator and the mapping function return the
14    // same address. The multiplication by the size of a pixel is because
15    // the generator returns addresses while the mapping functions returns
16    // the permutation
17    for(i= 0; i < gm.band_size(); i++)
18    {
19      if(gm.next() != offset + gm.q(i))
20      {
21        cout << "Error in assertion: " << i;
22      }
23    }

24    // Reinitialize the instance of the mapping-generator
25    gm.init();
26    // Check that the inverse address generator and the inverse mapping function
27    // return the same address.
28    for(i= 0; i < gm.band_size(); i++)
29    {
30      if(gm.inext() != offset + gm.qinv(i))
31      {
32        cout << "Error in inverse assertion: " << i ;
33      }
34    }

35    // Reinitialize the instance of the mapping-generator
36    gm.init();
37    // Check that the mapping function and the inverse mapping function
38    // are in fact inverses of each other.
39    for(i=0;i < gm.band_size(); i++)
40    {
41      int j;
42      if(i != gm.qinv(gm.q(i))) // Check that i=q^-1(q(i))
```

```
43          cout << "Inversion error: " << i;
44      }
45  #endif NOCHECK

46      // Allocate room for a band
47      pixel* image= new pixel[60*60*nn*mm];
48      GM gt(image, nn, mm); // Create an instance of a mapping with the base address image
49
50      for(i=0;i < gt.band_size(); i++)
51        *gt.next()= i;          // Write into the mapped band the pixel values
52                                // in order to know what pixel it is take as the
53                                // pixel value its sequential location in the band

54      // The following are the subset makeup of each abstraction.
55      // It is used to check that we can pull out an specific abstraction
56      // from the band. See equation (7).
57      cluster *c1[]= { &ca, &cb, &cc, &cd, &ce, &cf, &cg, &ch,
58                       &ci, &cj, &ck, &cspl, 0};
59      cluster *c2[]= { &ca, &cb, &cc, &cd, &ce, &cf, &cg, &ck, 0};
60      cluster *c3[]= { &ca, &cb, &cf, &cg, &ch, &ci, 0};
61      cluster *c4[]= { &ca, &cb, &cc, &cd, 0};
62      cluster *c5[]= { &ca, &cc, &ce, &cg, &ch, &cj, 0};
63      cluster *c6[]= { &ca, &cb, &cf, &cg, 0};
64      cluster *c10[]= { &ca, &cc, &ce, &cg, 0};

65      // Check if the subset construction is correct.
66      is_scaled_image(image, c1, 1, nn, mm);
67      is_scaled_image(image, c2, 2, nn, mm);
68      is_scaled_image(image, c3, 3, nn, mm);
69      is_scaled_image(image, c4, 4, nn, mm);
70      is_scaled_image(image, c5, 5, nn, mm);
71      is_scaled_image(image, c6, 6, nn, mm);
72      is_scaled_image(image, c10, 10, nn, mm);
73  }

74  // Check that the proper abstracted image can be extracted properly
75  void is_scaled_image(pixel* pimage,   cluster** pclusters, int divisor,
76                       int height, int width)
77  {
78    int i;
79    int image_size=0;
80    pixel* pp;
81    while(*pclusters)
82    {
83      cluster* pc= *pclusters;
84      image_size+= pc->size_of_set;
85      pp= pimage + width * height * pc->starting_location;
86      for(i=0; i < pc->size_of_set * height * width; i++)
```

```
87        {
88          if((*pp++ % divisor) != 0)
89            cerr << "out of order " << divisor ;
90        }
91        pclusters++;
92      }
93      if(image_size != 60*60/(divisor*divisor))
94        cerr << "Bad total size " << divisor;
95    }
```

CLAIMS

1. An image processing system comprising:

image acquisition means for acquiring pixels of a digital image;

permutation means for receiving said pixels subsequent to their acquisition by said image acquisition means and for ordering said pixels thereby to reorder said digital image; and

storage means for receiving and for storing said multiplicity of abstracted images.

2. An image processing system comprising:

image acquisition means for acquiring pixels of a digital image;

permutation means for receiving said pixels subsequent to their acquisition by said image acquisition means and for ordering said pixels thereby to reorder said digital image into a multiplicity of abstracted images at a reduction factor; and

storage means for receiving and for storing said multiplicity of abstracted images.

3. An image processing system comprising:

image acquisition means for acquiring pixels of a digital image;

permutation means for receiving said pixels subsequent to their acquisition by said image acquisition means, for ordering said pixels thereby to reorder said digital image into a plurality of abstracted images at a relatively large reduction factor and for combining said abstracted images into a multiplicity of sets of said abstracted images combinable into a plurality of abstracted images at a plurality of different relatively small reduction factors; and

storage means for receiving and for storing said digital image in said multiplicity of sets.

4. An image processing system according to claim 1 and wherein said permutation means comprise means for dividing said digital image into a plurality of bands comprising a predetermined number of rows of said digital image and means for operating on said plurality of bands in a band by band manner.

5. An image processing system according to claim 2 and wherein said permutation means comprise means for dividing said digital image into a plurality of bands comprising a predetermined number of rows of said digital image and means for operating on said plurality of bands in a band by band manner.

6. An image processing system according to claim 3 and wherein said permutation means comprise means for dividing said digital image into a plurality of bands comprising a predetermined number of rows of said digital image and means for operating on said plurality of bands in a band by band manner.

7. An image processing system according to claim 4 and wherein said means for operating operate separately on each band.

8. An image processing system according to claim 5 and wherein said means for operating operate separately on each band.

9. An image processing system according to claim 6 and wherein said means for operating operate separately on each band.

10. An image processing system according to claim 7 and wherein portions of said bands are written by said means for operating to said storage means when said portions of said band fill at least one file system block.

11. An image processing system according to claim 8 and wherein portions of said bands are written by said means for operating to said storage means when said portions of said band fill at least one file system block.

12. An image processing system according to claim 9 and wherein portions of said bands are written by said means for operating to said storage means when said portions of said band fill at least one file system block.

13. An image processing method including the steps of:

acquiring pixels of a digital image;

receiving said pixels subsequent to their acquisition in said step of acquiring and ordering said pixels thereby to reorder said digital image into a multiplicity of abstracted images; and

storing said multiplicity of abstracted images.

14. An image processing method including the steps of:

acquiring pixels of a digital image;

receiving said pixels subsequent to their acquisition in said step of acquiring and ordering said pixels thereby to reorder said digital image into a plurality of abstracted images at a relatively large reduction factor and combining said abstracted images into a multiplicity of sets of said abstracted images combinable into a plurality of abstracted images at a plurality of different relatively small reduction factors; and

storing said digital image in said multiplicity of sets.

15. An image processing method including the steps of:

acquiring pixels of a digital image;

ordering said pixels thereby to reorder said digital image into a multiplicity of abstracted images at a single reduction factor; and

storing said multiplicity of abstracted images.

16. An image transformation and rotation method including the steps of:

receiving an image of a given image size stored in a permuted format;

transforming said image to a tile format image via an inverse transformation of said permuted format to a sequential format and via a forward transformation from said sequential format to said tile format;

rotating said tile format image into a rotated tile format image by rotating the location of each tile of said image in said tile format, utilizing an amount of computer memory less than said image size;

retransforming said rotated tile format image to a rotated permuted format image via an inverse transformation from said tile format to said sequential format and via a forward transformation from said sequential format to said permuted format; and

storing said rotated permuted format image.

17. An image transformation and rotation method according to claim 16 and wherein said step of rotation includes the steps of:

receiving and buffering a first number of tiles of a source image to be rotated;

writing to a storage medium a second number of tiles of a destination image, said second number of tiles being different from said first number of tiles, which second number of tiles are fully defined by rotations of said first number of tiles of a source image; and repeating said steps of receiving and writing until said source image no longer contains tiles.

18. An image processing rotation method including the steps of:

receiving and buffering a first number of tiles of a source image to be rotated;

writing to a storage medium a second number of tiles of a destination image, said second number of tiles being different from said first number of tiles, which second number of tiles are fully defined by rotations of said first number of tiles of a source image; and

repeating said steps of receiving and writing until said source image no longer contains tiles.

19. An image processing system comprising:

image acquisition means for acquiring pixels of a digital image;

means for dividing said digital image into a plurality of bands each comprising a predetermined number of rows; and

permutation means for operating on said plurality of bands in a band by band manner and for reordering the location of said pixels;

20. A system according to claim 19 and also comprising storage means for receiving and for storing said bands in a band by band manner.

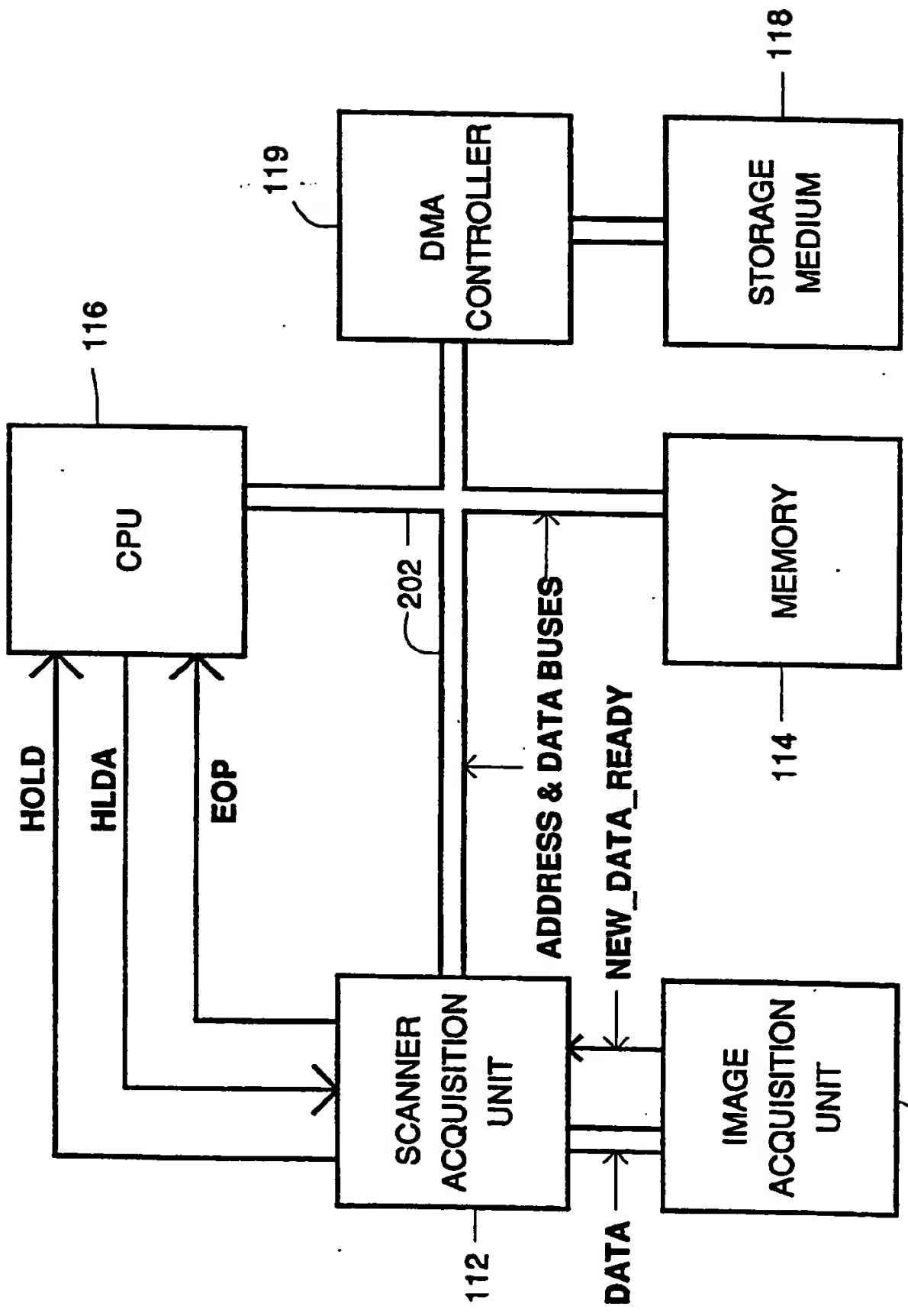
118
119
DMA CONTROLLER
STORAGE MEDIUM
116
CPU
MEMORY
202
ADDRESS & DATA BUS
NEW_DATA_READY
HOLD
HLDA
EOP
114
SCANNER ACQUISITION UNIT
IMAGE ACQUISITION UNIT
110
112
DATA

Fig. 1.

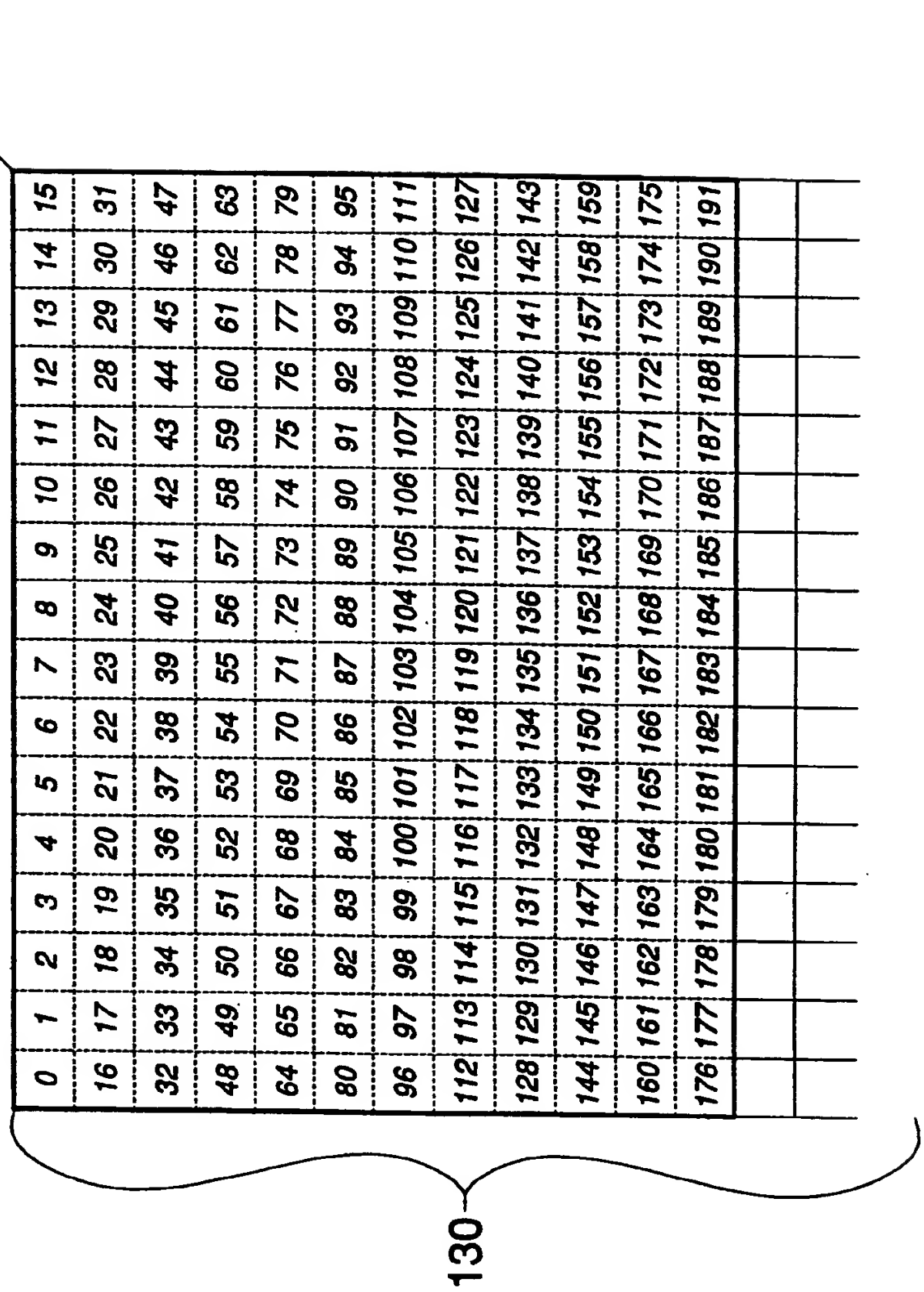
131
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
130

Fig. 2A.

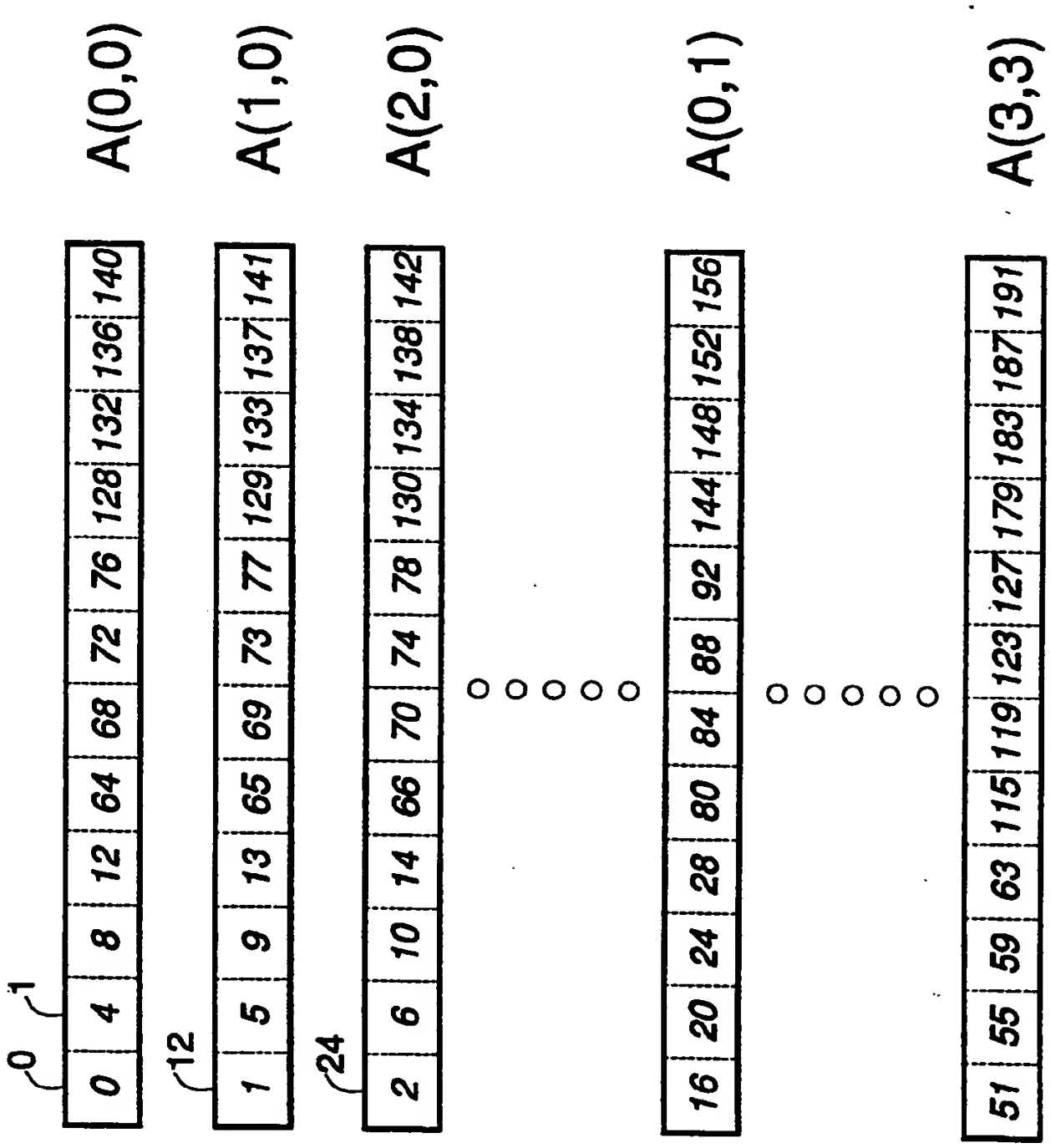
A(0,0)
A(1,0)
A(2,0)
A(0,1)
A(3,3)
0
1
12
24
0 4 8 12 64 68 72 76 128
1 5 9 13 65 69 73 77 129
2 6 10 14 66 70 74 78 130
16 20 24 28 80 84 88 92 144
51 55 59 63 115 119 123 127 179 183 187 191

Fig. 2B.

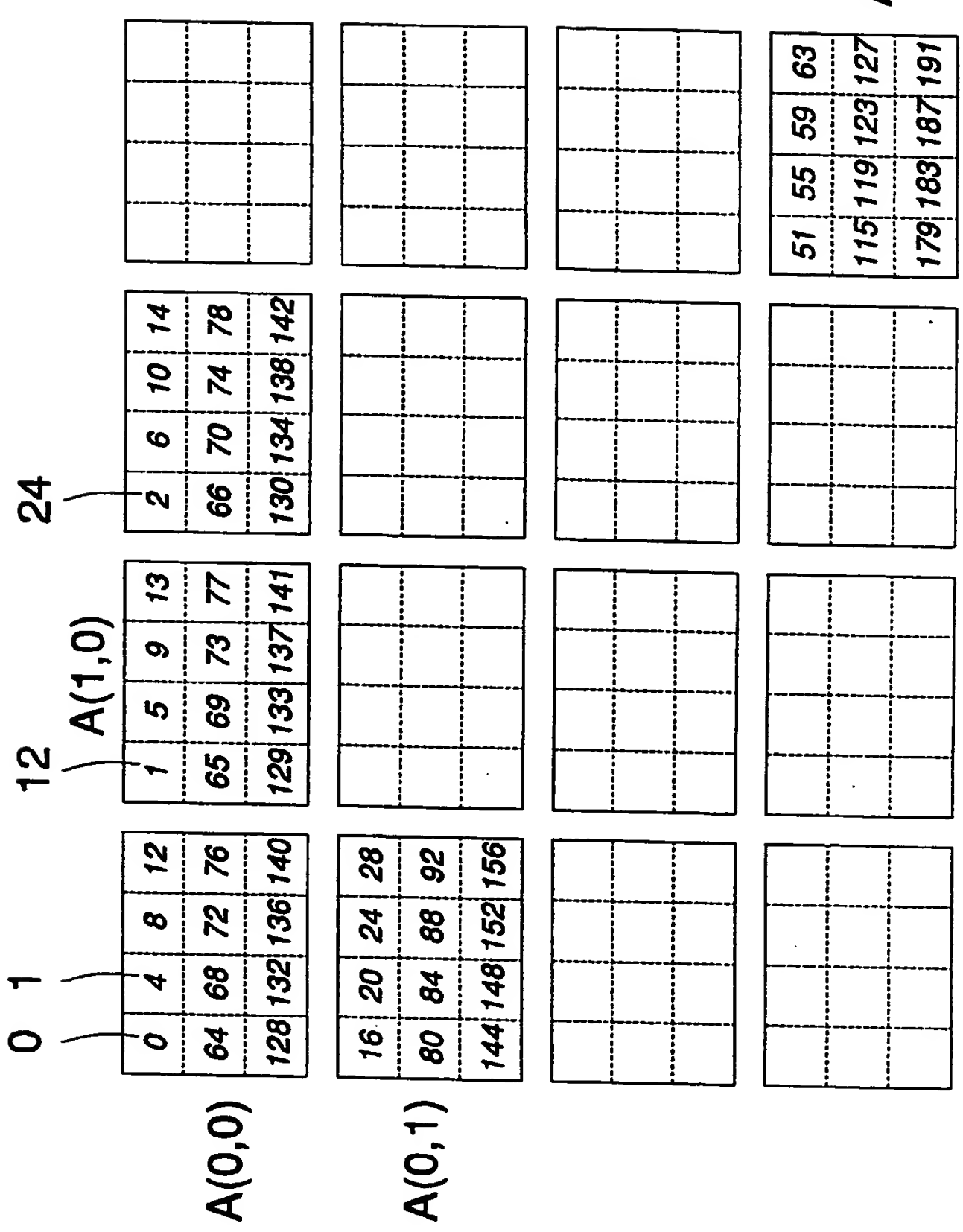
A(3,3)
A(0,0)
A(0,1)
A(1,0)
0
1
12
24
0 4 8 12 64 68 72 76 128 132 136 140
16 20 24 28 80 84 88 92 144 148 152 156
1 5 9 13 65 69 73 77 129 133 137 141
2 6 10 14 66 70 74 78 130 134 138 142
51 55 59 63 115 119 123 127 179 183 187 191

Fig. 2C.

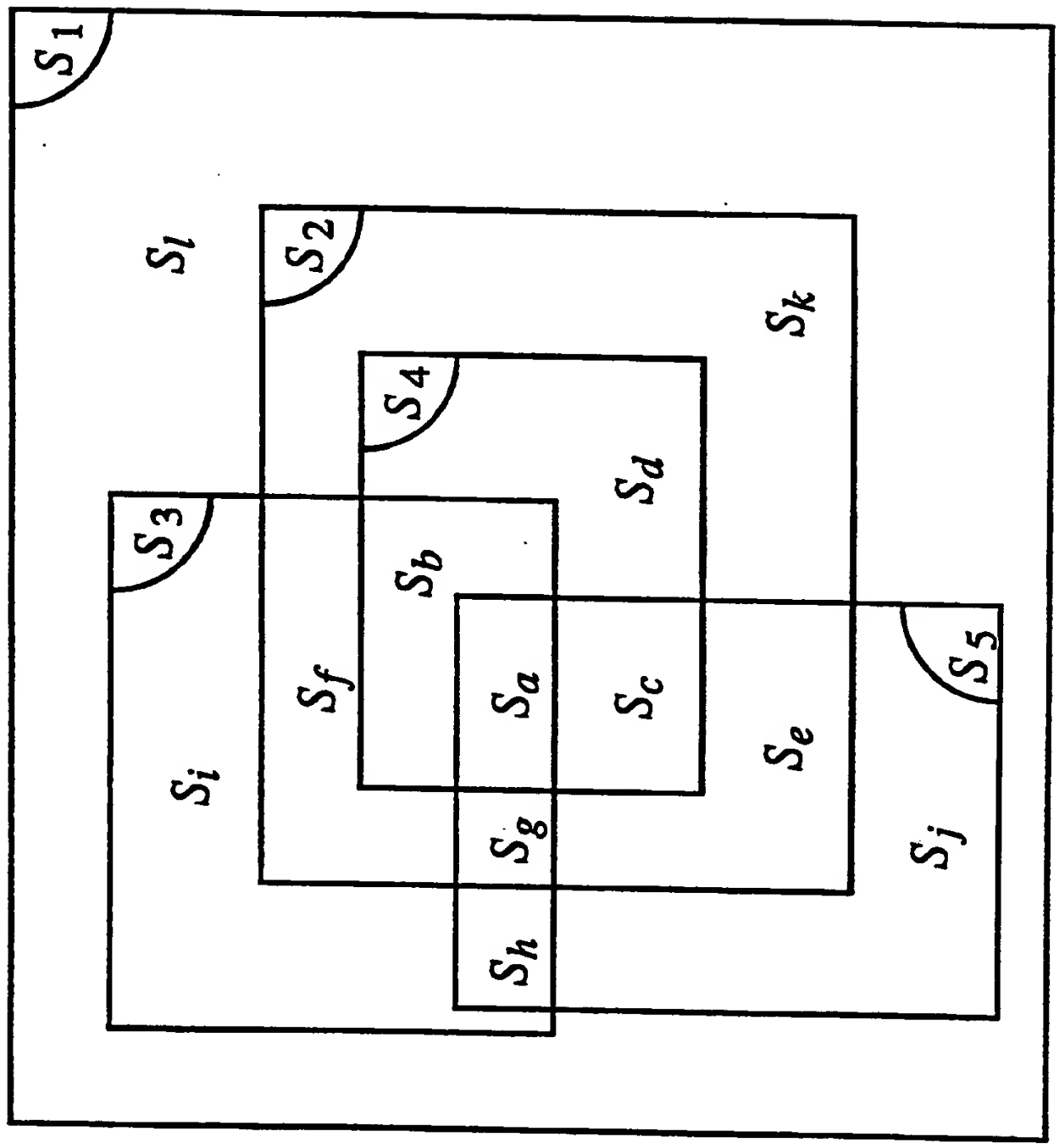

Fig. 3.

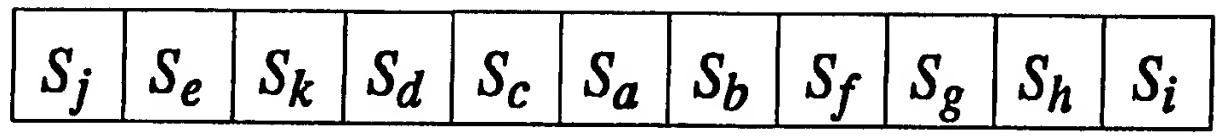

Fig. 4A

S_2

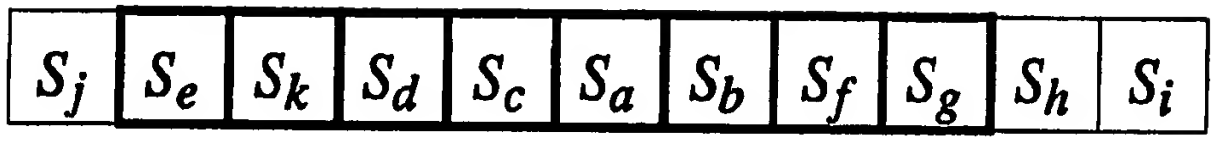

Fig. 4B

S_3

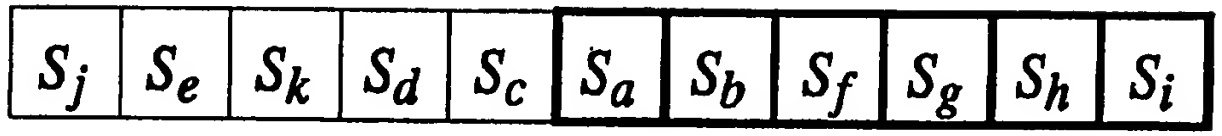

Fig. 4C

S_4

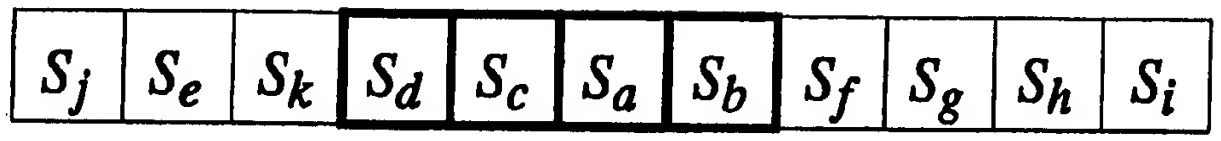

Fig. 4D

S_5

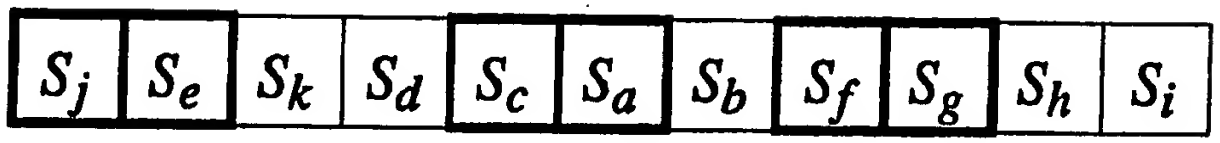

Fig. 4E

S_6

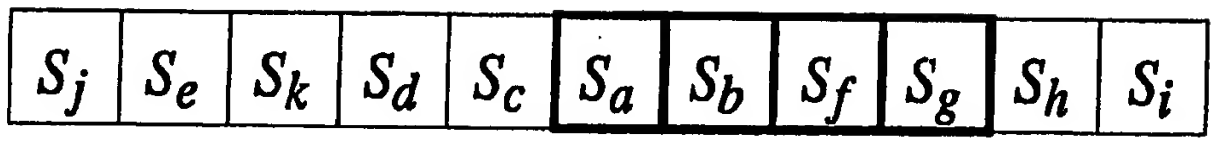

Fig. 4F

1/10/2005, EAST Version: 2.0.1.4

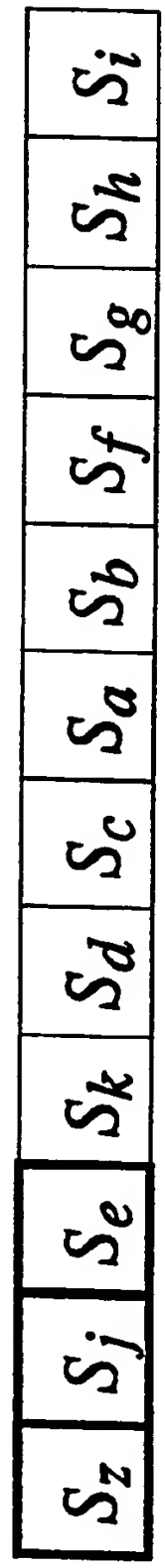

Fig. 5.

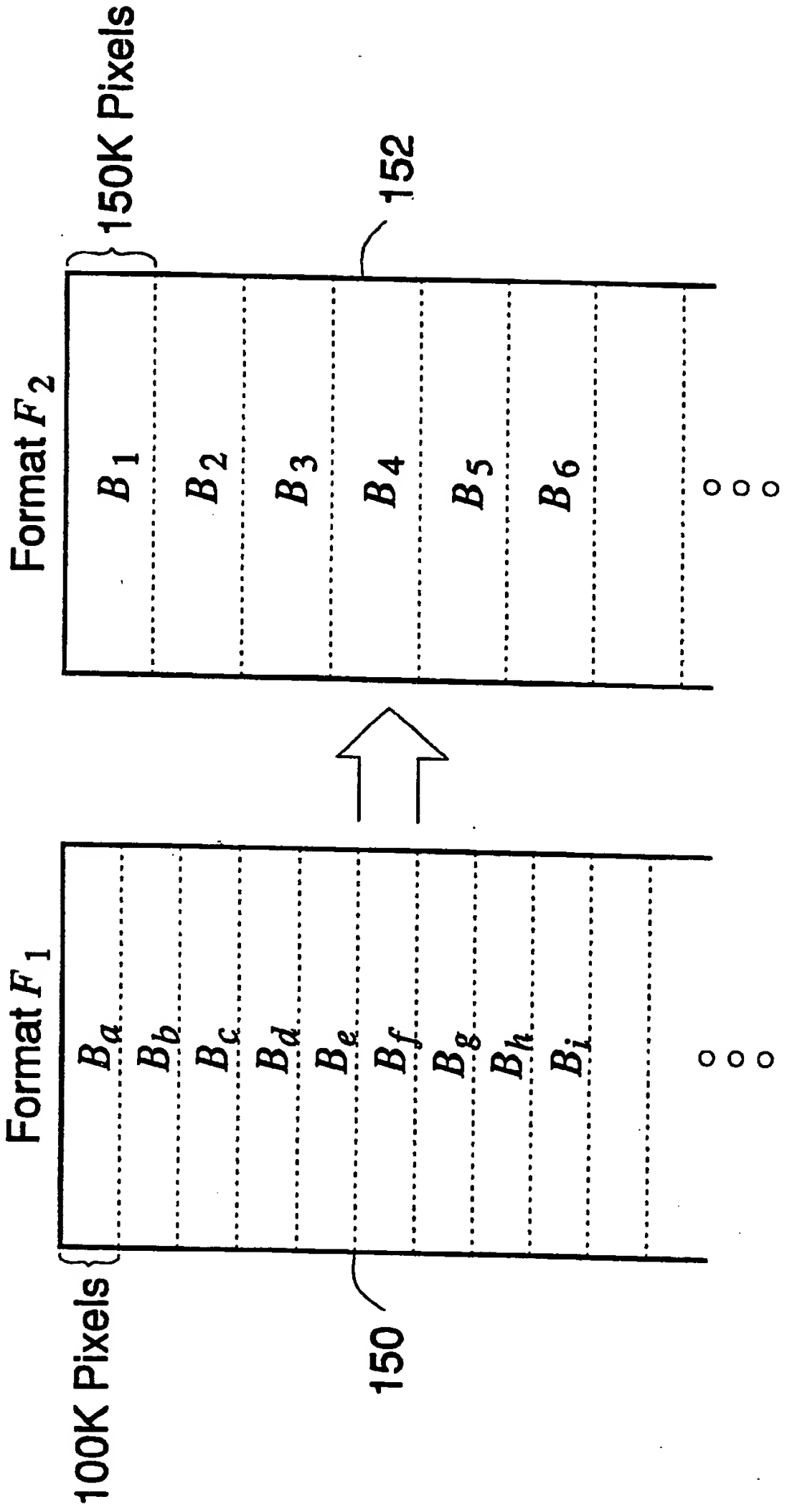
Format F_2
150K Pixels
152
B_1
B_2
B_3
B_4
B_5
B_6
Format F_1
100K Pixels
150
B_a
B_b
B_c
B_d
B_e
B_f
B_g
B_h
B_i

Fig. 6.

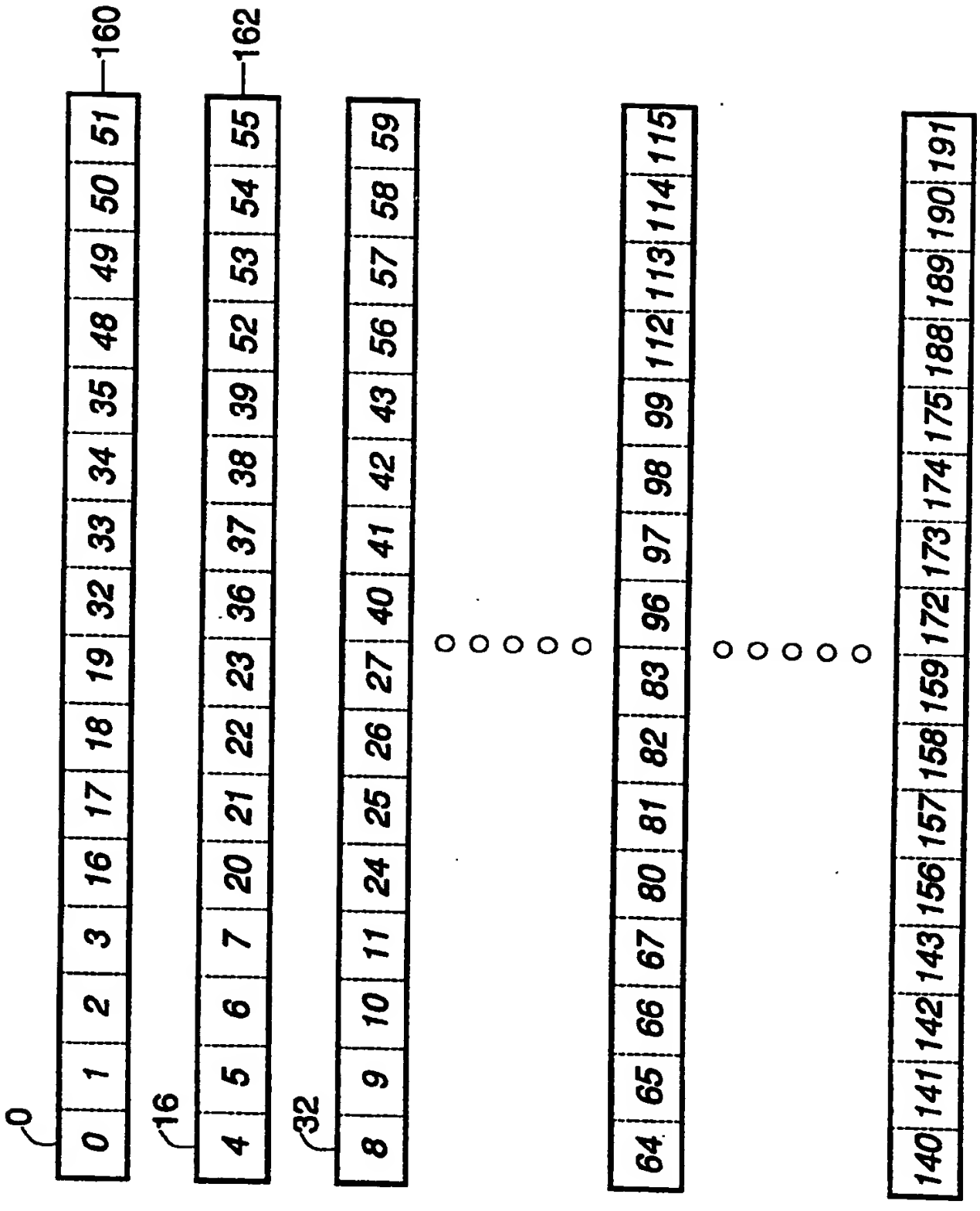
160
162
0
16
32
0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51
4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55
8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59
64 65 66 67 80 81 82 83 96 97 98 99 112 113 114 115
140 141 142 143 156 157 158 159 172 173 174 175 188 189 190 191

Fig. 7.

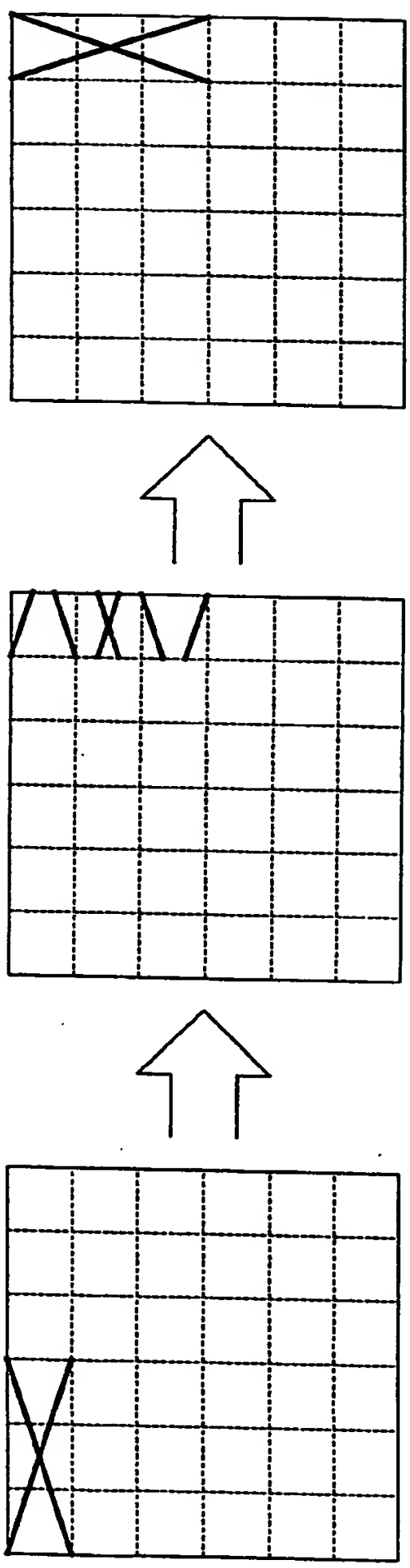

Fig. 8.

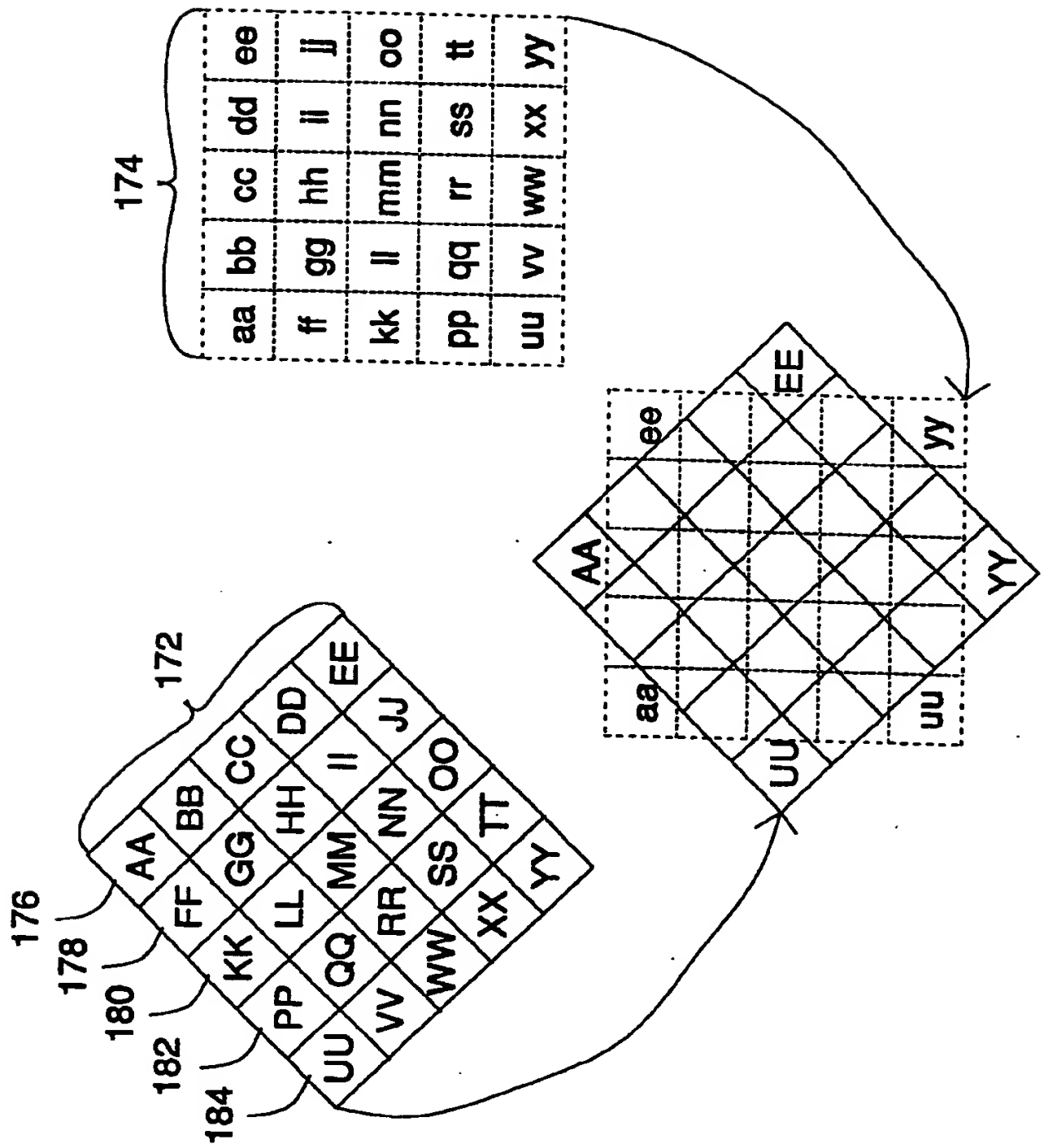
174
aa ff kk pp uu
bb gg ll qq vv
cc hh mm rr ww
dd ii nn ss xx
ee jj oo tt yy
EE
ee yy
AA YY
aa uu
UU
172
AA BB CC DD EE
FF GG HH II JJ
KK LL MM NN OO
PP QQ RR SS TT
UU VV WW XX YY
176
178
180
182
184

Fig. 9.

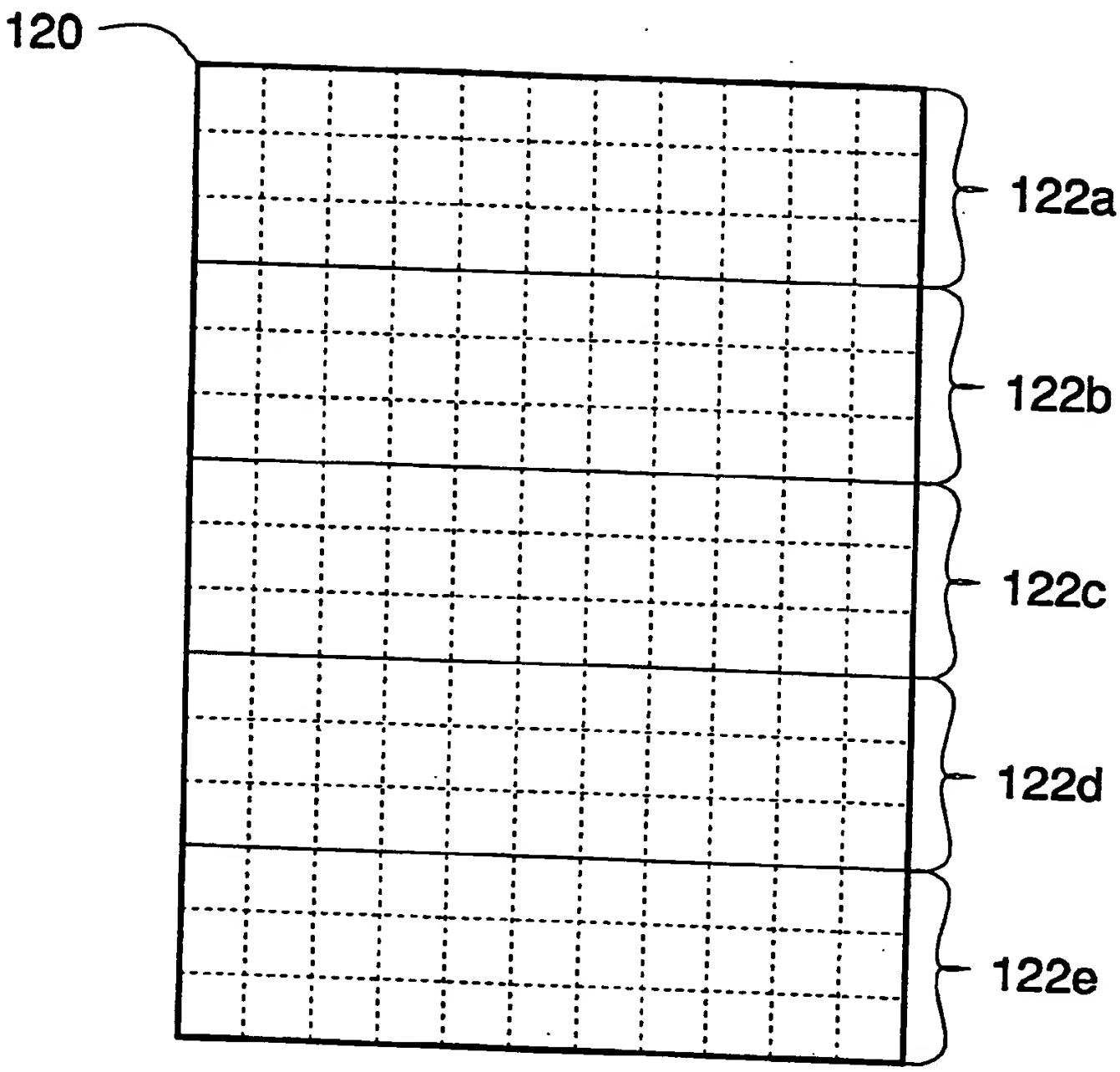

Fig. 10.